International Telegraph and Telephone
Consultative Committee
(CCITT)
Period 1989-1992

Doc. #487
March 1989
Original: English

Question: XV/4 Specialists Group

STUDY GROUP XV - CONTRIBUTION

Source: USA

Title: Information On A Means For Implementation Of Reed-Solomon Error Correction For p x 64 kbit/s Video Telephony

1. Overview

In this document, an implementation of Reed-Solomon error correction is presented in detail. In addition to the high performance provided by Reed-Solomon error correction, it is shown here that it can be implemented very simply, either with off-the-shelf hardware or using custom integrated circuits.

The attached documentation shows a complete hardware and software implementation of Reed-Solomon error correction. The error correction procedure may be divided into three independent tasks:

1. Checksum generation (transmitter)
2. Syndrome generation (receiver)
3. Error corrector (receiver)

In the implementation described, the checksum and syndrome generation has been implemented in dedicated hardware using off-the-shelf components. The error corrector has been implemented in software.

The attached documentation consists of the following:

1. Functional block diagram of the checksum generator.
2. Functional block diagram of the syndrome generator.
3. Hardware block diagram of checksum/syndrome generator.
4. Hardware schematic of checksum/syndrome generator.
5. Firmware for programmable array logic (PAL) devices.
6. Firmware for EPROM look-up table
7. Software source listing for the error corrector.

The implementation presented assumes a block length of 255 octets. However, the design would remain basically unchanged if a smaller truncated block is used.

2. Checksum and Syndrome Generator

The discrete implementation of a two-channel checksum and syndrome generator consists of the following components:

Look-up table (2k x 8 EPROM)
LUT register (1 MSI)
XOR accumulator (1 PAL)
Checksum and Syndrome registers (32 x 8 bit RAM)
Sequencer (2 PALs, 1 other MSI)

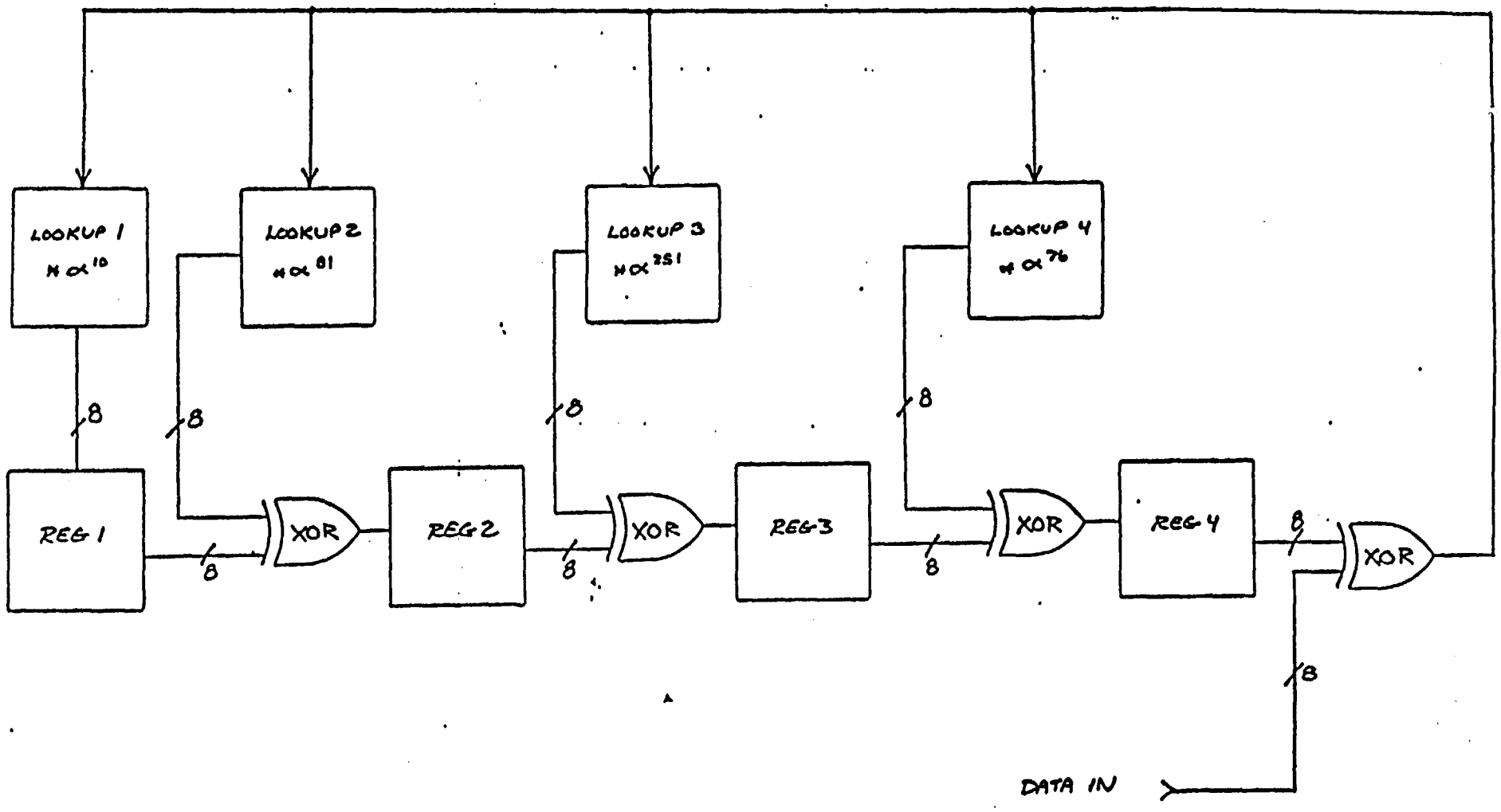
This constitutes a total of 7 ICs. The bus interface consists of another 3 ICs in this implementation. A 4 MHz clock rate was used, which is sufficient to handle encoding and decoding at 1 Mbit/s. For a 2 Mbit/s rate, an 8 MHz clock would be required.

This design has also been implemented as part of a custom gate-array chip. In this implementation, the design consists of approximately 1200 gates plus the use of 2k x 8 of external RAM to hold the look-up tables. The maximum throughput in this implementation is approximately 5 Mbit/s.

3. Error Corrector

The error corrector has been implemented in software. The software, written in C, was designed to operate on an Intel 80286 microprocessor. In this implementation, the software runs on the same shared processor as the communication and system control software used for operation of the video codec.

The worst-case software loading is about 150 instructions per block (for double errors). For the maximum block length (2040 bits), this corresponds to 0.074 instructions per bit; for a truncated block length of 128 octets (1024 bits), it corresponds to 0.146 instructions per bit. At 2 Mbit/s (1856 kbit/s video), this corresponds to 136 and 272 k inst/s, respectively. This can easily be handled by a microcontroller or a portion of a microprocessor.



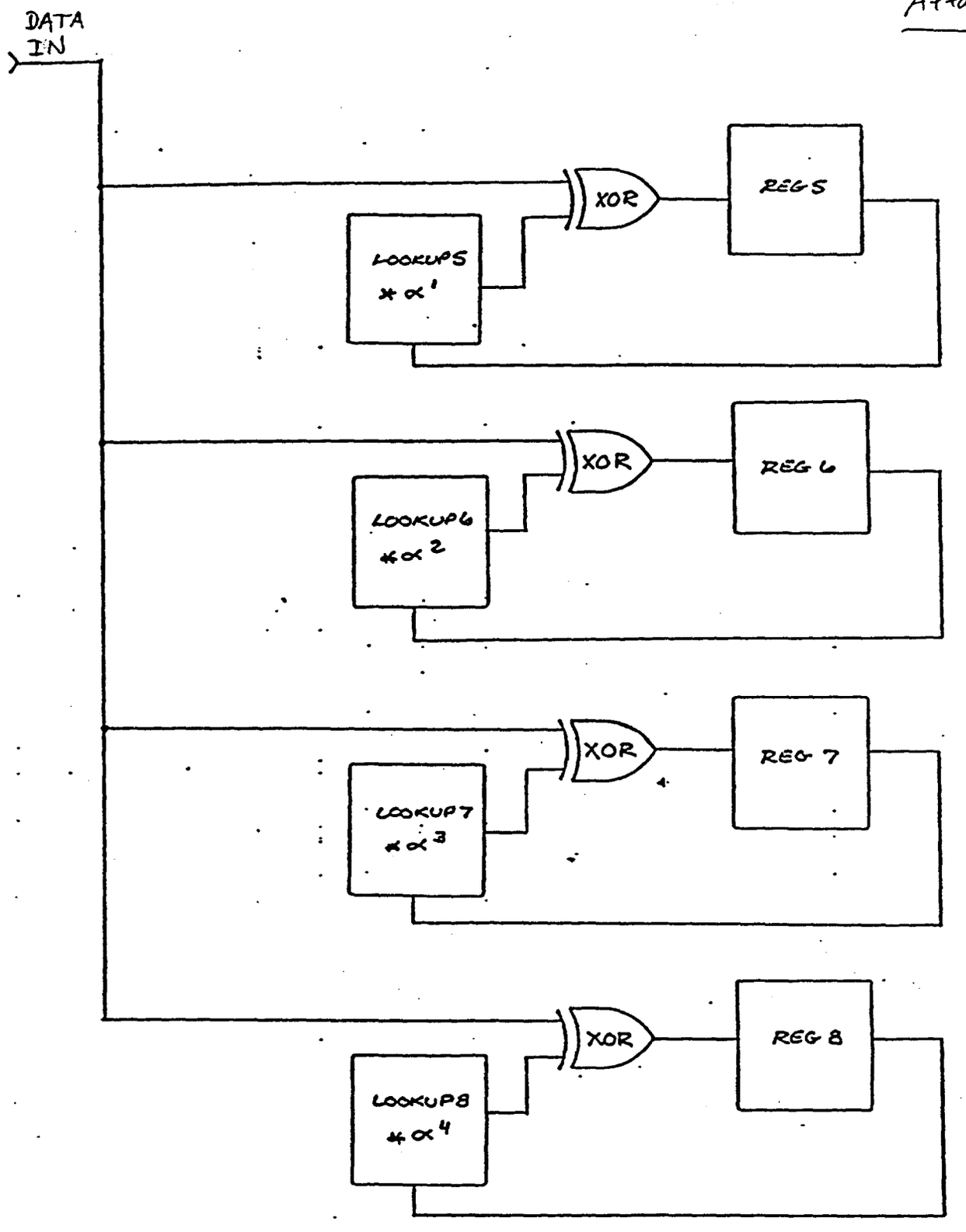
CHECKSUM GENERATING SCHEME

Attachment 1

M. GREIM
4-3-87

SYNDROME GENERATION SCHEME

Attachment 2.

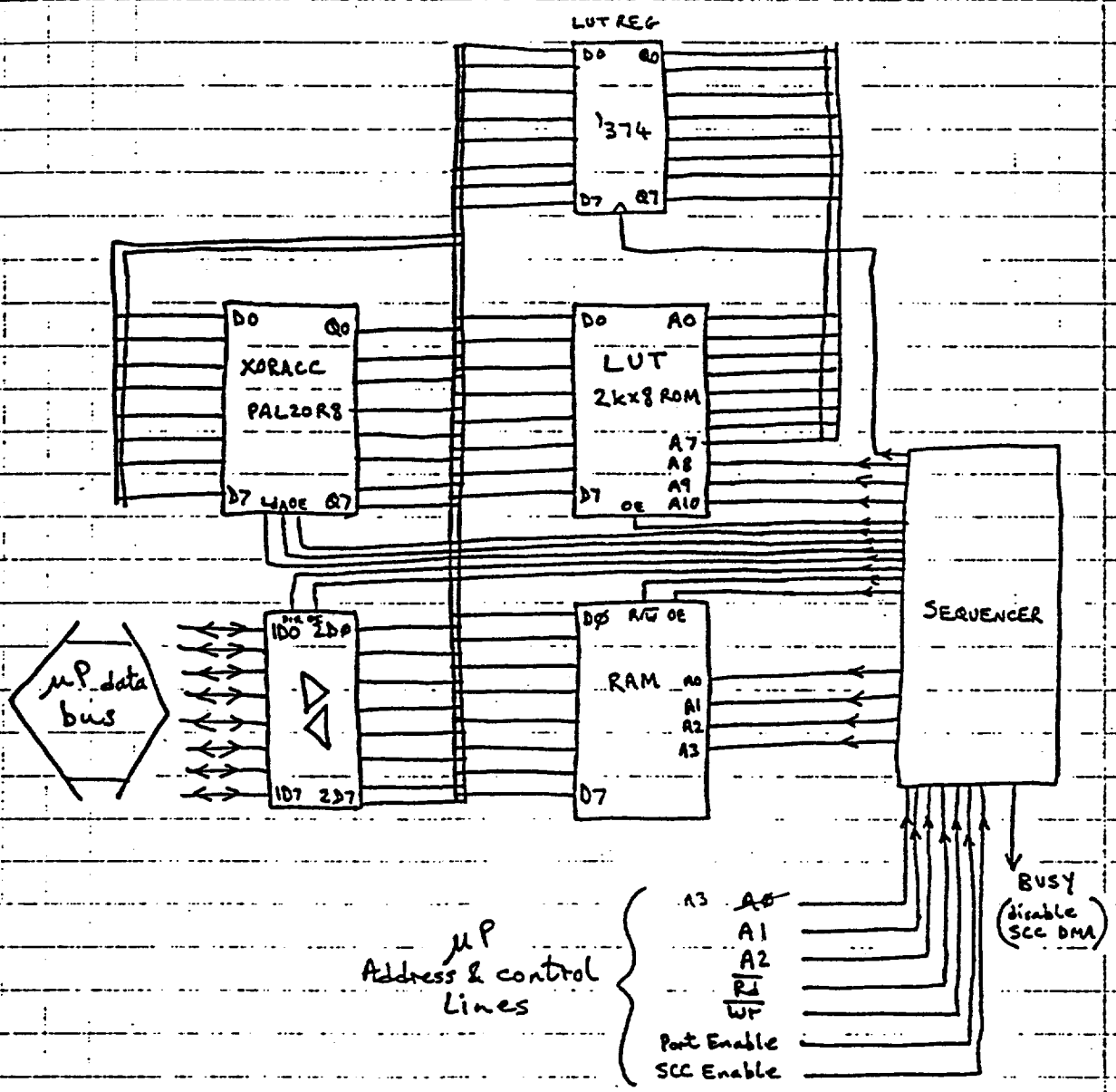


M. GREIM
4-3-87

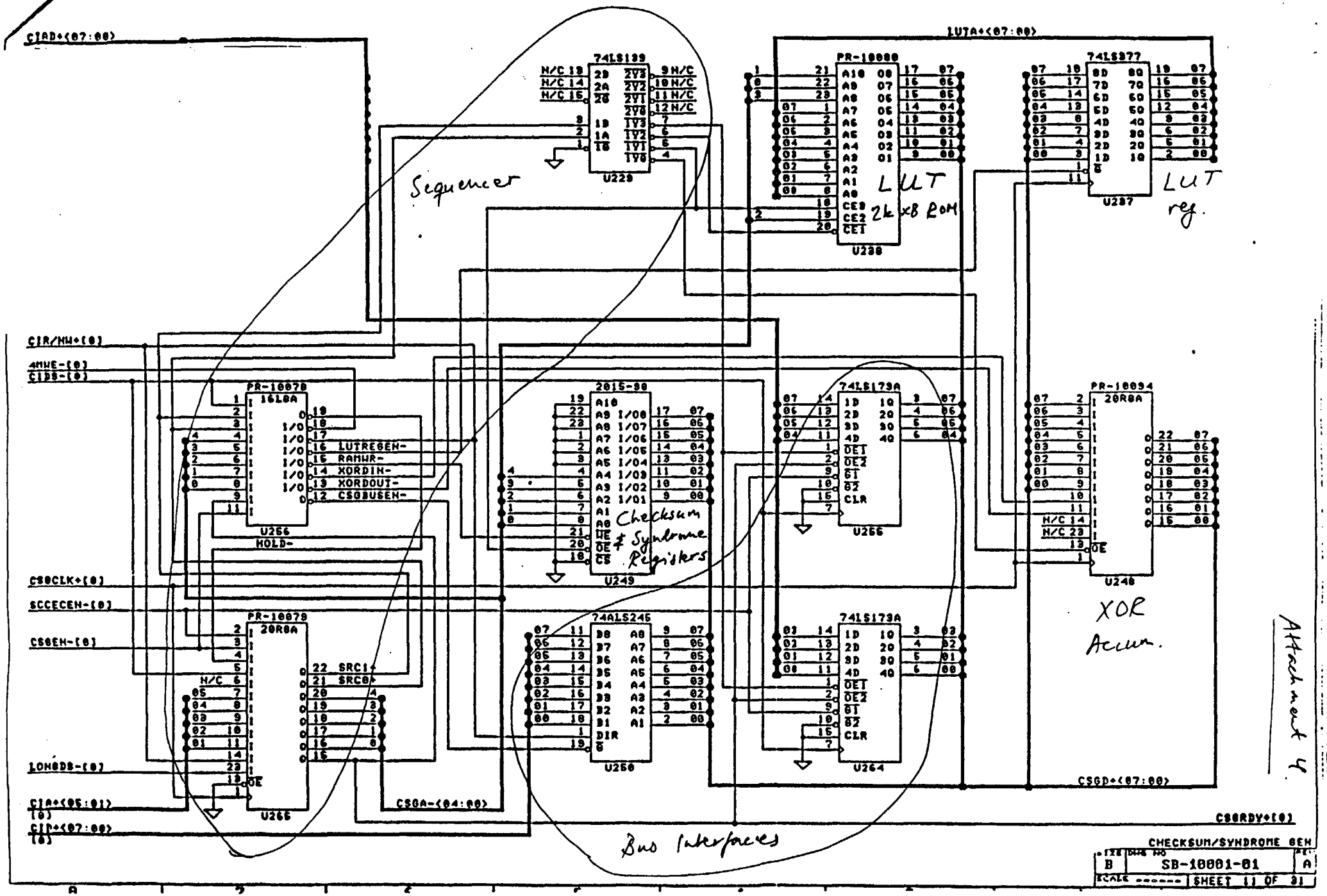
850614.2 CRC

Attachment 3.

Check Symbol & Syndrome generation hardware



3Xo



Attachment 4

```
Module  CSG4_Decoder

Flag    '-r3'

Title   'CHANNEL INTERFACE, Checsum / Syndrome Generator Decoder
        Copyright (c) PicTel Corporation, 1986
        Craig Clapp   30 April 1986'


"Declarations

PR10078  device 'P16L8';


"Inputs

        CSGRDY,_4MWE                            PIN 9,18;
        _CIDS,CIRNW,_CSGEN                      PIN 1,17,11;
        Src0,Src1                               PIN 3,2;
        _Addr0,_Addr1,_Addr2,_Addr3,_Addr4      PIN 8,7,6,5,4;


"Outputs

        _RAMWrite,_LUTRegEn                     PIN 15,16;
        _XorDOut,_XorDIn                        PIN 13,14;
        _CSGBusEn                               PIN 12;
        _Hold                                   PIN 19;


"Define constants

        Source   = [CSGRDY,Src1,Src0];
        Count    = [!_Addr2,!_Addr1,!_Addr0];
        Max      = 3;     "Highest numbered Checksum or Syndrome byte
        Checksum = (!_Addr3 == 0);
        Syndrome = (!_Addr3 == 1);

        Ready   = ^B111;
        Listen  = ^B110;
        RAM     = ^B001;
        CPU     = ^B011;
        LUT     = ^B010;
        XORACC  = ^B000;
        ReadRAM = ^B101;
        ClrRAM  = ^B100;


        LoadXor = (Source == LUT) & Checksum
                # (Source == CPU) & Syndrome
                # (Source == CPU) & Checksum & (Count == 0);

        XOR     = (Source == RAM) & Checksum
                # (Source == LUT) & Syndrome;

        ClrXor  = (Source == Ready  )
                # (Source == Listen )
                # (Source == ReadRAM)
```

Attachment 5.
(9 p.)

```
                # (Source == ClrRAM );

        HoldXor = (Source == CPU) & Checksum & (Count != 0)
                # (Source == RAM) & Syndrome
                # (Source == XORACC);

Equations

        !_LUTRegEn = (Source == RAM    ) & Syndrome
                   # (Source == XORACC) & Checksum & (Count == -1);

        !_RAMWrite = (Source == ClrRAM) & !_4MWE
                   # (Source == XORACC) & Syndrome & !_4MWE
                   # (Source == XORACC) & Checksum & (Count != -1 ) & !_4MWE
                   # (Source == LUT    ) & Checksum & (Count == Max) & !_4MWE
                   # (Source == Ready ) & !_CSGEN & !_CIDS & !CIRNW;

        !_XorDOut  = HoldXor # XOR;

        !_XorDIn   = LoadXor # XOR;

        !_CSGBusEn = (Source == ReadRAM) & !_CSGEN &  CIRNW & !_CIDS
                   # (Source == Ready  ) & !_CSGEN & !CIRNW;

        !_Hold  = ((Source == RAM    ) # (Source ==LUT)) & Syndrome
                #  (Source == CPU    ) # (Source == ReadRAM)
                #  (Source == ClrRAM) # (Source == Listen );


End CSG4_Decoder;
```

```
Module  CSG4_Sequencer

Flag    '-r3'

Title   'CHANNEL INTERFACE, Checksum / Syndrome Generator sequencer
        Copyright (c) PicTel Corporation, 1986
        Craig Clapp   30 April 1986'


"Declarations

PR10079  device 'P20R8';


"Inputs

        Clk, _OE                                PIN 1,13;
        _SCCECEN, _CSGEN                        PIN 2,3;
        _CIDS,CIRNW,_LONGDS                     PIN 5,14,23;
        CIA01,CIA02,CIA03,CIA04,CIA05           PIN 11,10,9,8,7;
        _Hold                                   PIN 4;


"Outputs

        CSGRDY                                  PIN 15;
        Src0,Src1                               PIN 21,22;
        _Addr0,_Addr1,_Addr2,_Addr3,_Addr4      PIN 16,17,18,19,20;


"Define constants

        X = .X.;
        H = (1==1);
        L = 0;

        CIA     = [CIA05,CIA04,CIA03,CIA02,CIA01];
        _DS     = [_LONGDS,_CIDS];
        Source  = [CSGRDY,Src1,Src0];
        Count   = [!_Addr2,!_Addr1,!_Addr0];
        Max     = 3;    "Highest numbered Checksum or Syndrome byte
        Checksum =  _Addr3;
        Syndrome = !_Addr3;

        Ready   = ^B111;
        Listen  = ^B110;
        RAM     = ^B001;
        CPU     = ^B011;
        LUT     = ^B010;
        XORACC  = ^B000;
        ReadRAM = ^B101;
        ClrRAM  = ^B100;

        Up      = ((Source == XORACC) # (Source == LUT)) & _Hold;

        Down    = ((Source == CPU   ) # (Source == RAM)) & _Hold;

        Load    =  _SCCECEN & (Source == Ready);
```

```
        Clear   = !_SCCECEN & (Source =- Ready);

"       !_Hold  = ((Source == RAM    ) # (Source == LUT     )) & Syndrome
"                 # (Source == CPU    ) # (Source == ReadRAM)
"                 # (Source == ClrRAM) # (Source == Listen );
" Generate '_Hold' as an input pin to reduce OR terms in Addr equations


STATE_DIAGRAM Source

        State Ready:    Case    !_CIDS & !_SCCECEN              :Listen;
                                !_CIDS & !_CSGEN & CIRNW        :ReadRAM;
                        !(!_CIDS & (!_SCCECEN # !_CSGEN & CIRNW))      :Ready;
                        Endcase;

        State Listen:   Case    !_LONGDS                :Listen;
                                 _LONGDS & Checksum     :CPU;
                                 _LONGDS & Syndrome     :RAM;
                        Endcase;

        State RAM:      Case    Checksum        :XORACC;
                                Syndrome        :CPU;
                        Endcase;

        State CPU:      Case    Checksum        :RAM;
                                Syndrome        :LUT;
                        Endcase;

        State LUT:      Case    Checksum & (Count != Max)       :CPU;
                                Checksum & (Count == Max)       :Ready;
                                Syndrome                        :XORACC;
                        Endcase;

        State XORACC:   Case    Checksum                        :LUT;
                                Syndrome & (Count != Max)       :RAM;
                                Syndrome & (Count == Max)       :Ready;
                        Endcase;

        State ReadRAM:  If !_LONGDS == 0 Then ClrRAM Else ReadRAM;

        State ClrRAM:   Goto Ready;


EQUATIONS

!_Addr0 := !_Addr0 & !_Hold
         #  _CIA01 & Load
         #  _Addr0 & Up
         #  _Addr0 & Down;
 "Default 'Clear' state is  _Addr0 = 1  when _Hold & !Load & !Up & !Down

!_Addr1 := !_Addr1 & !_Hold
         #  _CIA02 &  _Load
         # !_Addr1 &  _Addr0 & Up
         #  _Addr1 & !_Addr0 & Up
         # !_Addr1 & !_Addr0 & Down
         #  _Addr1 &  _Addr0 & Down;
```

```
 "Default 'Clear' state is  _Addr1 = 1  when _Hold & !Load & !Up & !Down

!_Addr2 := !_Addr2 & !_Hold
         #   CIA03 & Load
         # !_Addr2 & ( _Addr1 #  _Addr0) & Up
         #  _Addr2 &  !_Addr1 & !_Addr0  & Up
         # !_Addr2 & (!_Addr1 # !_Addr0) & Down
         #  _Addr2 &   _Addr1 &  _Addr0  & Down;
 "Default 'Clear' state is  _Addr2 = 1  when _Hold & !Load & !Up & !Down

!_Addr3 := CIRNW & Clear # CIA04 & Load # !_Addr3 & !Clear & !Load;

!_Addr4 := CIA05 & Clear # CIA05 & Load # !_Addr4 & !Clear & !Load;


Test_Vectors

([Clk,_OE,_DS,CIRNW,_SCCECEN,_CSGEN,CIA,_Hold]->[_Addr4,_Addr3,Count,Source])

"Initialize outputs

 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  1  ]->[   X  ,   X  ,  X  ,  X   ];
 [.C., 0 , H ,  1  ,    1   ,   1  , 0 ,  0  ]->[   X  ,   X  ,  X  ,Ready ];

"Test CPU Access
```

```
"[Clk,_OE,_DS,CIRNW,_SCCECEN,_CSGEN,C:  _Hold]->[_Addr4,_Addr3,Count,Source]

 [.C., 0 , L ,  0  ,    1   ,   0  ,21 ,  1  ]->[   0  ,   1  ,  5  ,Ready ];
 [.C., 0 , H ,  1  ,    1   ,   0  ,10 ,  1  ]->[   1  ,   0  ,  2  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 0 ,  1  ]->[   1  ,   1  ,  0  ,ReadRAM];
 [.C., 0 , L ,  1  ,    1   ,   0  , 0 ,  0  ]->[   1  ,   1  ,  0  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , 0 ,  0  ]->[   1  ,   1  ,  0  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , 0 ,  0  ]->[   1  ,   1  ,  0  ,Ready ];
 [.C., 0 , H ,  1  ,    1   ,   1  , 0 ,  1  ]->[   1  ,   1  ,  0  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 1 ,  1  ]->[   1  ,   1  ,  1  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   1  ,   1  ,  1  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   1  ,   1  ,  1  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 2 ,  1  ]->[   1  ,   1  ,  2  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   1  ,   1  ,  2  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   1  ,   1  ,  2  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 3 ,  1  ]->[   1  ,   1  ,  3  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   1  ,   1  ,  3  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   1  ,   1  ,  3  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 4 ,  1  ]->[   1  ,   1  ,  4  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   1  ,   1  ,  4  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   1  ,   1  ,  4  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  , 8 ,  1  ]->[   1  ,   0  ,  0  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   1  ,   0  ,  0  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   1  ,   0  ,  0  ,Ready ];
 [.C., 0 , L ,  1  ,    1   ,   0  ,16 ,  1  ]->[   0  ,   1  ,  0  ,ReadRAM];
 [.C., 0 , H ,  1  ,    1   ,   0  , X ,  0  ]->[   0  ,   1  ,  0  ,ClrRAM];
 [.C., 0 , H ,  1  ,    1   ,   1  , X ,  0  ]->[   0  ,   1  ,  0  ,Ready ];

"Test Checksum Sequence
"[Clk,_OE,_DS,CIRNW,_SCCECEN,_CSGEN,CIA,_Hold]->[_Addr4,_Addr3,Count,Source]

 [.C., 0 , H ,  X  ,    1   ,   1  ,14 ,  1  ]->[   1  ,   0  ,  6  ,Ready ];
 [.C., 0 , H ,  0  ,    0   ,   1  ,14 ,  1  ]->[   1  ,   1  ,  0  ,Ready ];
 [.C., 0 , H ,  X  ,    1   ,   1  ,14 ,  1  ]->[   1  ,   0  ,  6  ,Ready ];
 [.C., 0 , L ,  0  ,    0   ,   1  ,14 ,  1  ]->[   1  ,   1  ,  0  ,Listen];
 [.C., 0 , L ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  0  ,Listen];
 [.C., 0 , H ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  0  , CPU  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  0  ]->[   1  ,   1  ,  0  , RAM  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  , -1  ,XORACC];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  0  , LUT  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  1  , CPU  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  0  ]->[   1  ,   1  ,  1  , RAM  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  0  ,XORACC];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  1  , LUT  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  2  , CPU  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  0  ]->[   1  ,   1  ,  2  , RAM  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  1  ,XORACC];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  2  , LUT  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  3  , CPU  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  0  ]->[   1  ,   1  ,  3  , RAM  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  2  ,XORACC];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  3  , LUT  ];
 [.C., 0 , X ,  X  ,    X   ,   X  , X ,  1  ]->[   1  ,   1  ,  4  ,Ready ];

"Test Syndrome Sequence
"[Clk,_OE,_DS,CIRNW,_SCCECEN,_CSGEN,CIA,_Hold]->[_Addr4,_Addr3,Count,Source]

 [.C., 0 , H ,  X  ,    1   ,   1  ,21 ,  1  ]->[   0  ,   1  ,  5  ,Ready ];
```

```
[.C., 0 , H , 1  ,    0   ,   1  ,21    1  ]->[   0  ,   0  ,  0  ,Ready ];
[.C., 0 , H , X  ,    1   ,   1  ,21  , 1  ]->[   0  ,   1  ,  5  ,Ready ];
[.C., 0 , L , 1  ,    0   ,   1  ,21  , 1  ]->[   0  ,   0  ,  0  ,Listen];
[.C., 0 , L , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  0  ,Listen];
[.C., 0 , H , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  0  , RAM  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  0  , CPU  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  0  , LUT  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  0  ,XORACC];
[.C., 0 , X , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  1  , RAM  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  1  , CPU  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  1  , LUT  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  1  ,XORACC];
[.C., 0 , X , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  2  , RAM  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  2  , CPU  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  2  , LUT  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  2  ,XORACC];
[.C., 0 , X , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  3  , RAM  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  3  , CPU  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  3  , LUT  ];
[.C., 0 , X , X  ,    X   ,   X  , X  , 0  ]->[   0  ,   0  ,  3  ,XORACC];
[.C., 0 , X , X  ,    X   ,   X  , X  , 1  ]->[   0  ,   0  ,  4  ,Ready ];

End CSG4_Sequencer;
```

```
Module  XOR_Acc

Flag    '-r3'

Title   'CHANNEL INTERFACE, Exclusive-OR Accumulator
        Copyright (c) PicTel Corporation, 1986
        Craig Clapp   1 May 1986'


"Declarations

PR10094  device 'P20R8';


"Inputs

        Clk,_OE                         PIN 1,13;
        D7,D6,D5,D4,D3,D2,D1,D0         PIN 2,3,4,5,6,7,8,9;
        _XorDIn,_XorDOut                PIN 10,11;

        DataIn = [D7,D6,D5,D4,D3,D2,D1,D0];


"Outputs

        Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0          PIN 22,21,20,19,18,17,16,15;

        DataOut = [Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0];


Equations

        DataOut := DataOut & (!_XorDOut == 1) $ DataIn & (!_XorDIn == 1);


Test_Vectors

      ( [ Clk,_OE,DataIn,_XorDOut,_XorDIn] -> DataOut )

        [ .C., 0 , ^H93 ,    1    ,    1    ] ->  ^H00    ; "Clear
        [ .C., 0 , ^H93 ,    1    ,    0    ] ->  ^H93    ; "Load
        [ .C., 0 , ^H7C ,    0    ,    1    ] ->  ^H93    ; "Hold
        [ .C., 0 , ^H7C ,    0    ,    0    ] ->  ^HEF    ; "XOR

        [ .C., 0 , ^H00 ,    1    ,    0    ] ->  ^H00    ;
        [ .C., 0 , ^H04 ,    1    ,    0    ] ->  ^H04    ;
        [ .C., 0 , ^H08 ,    1    ,    0    ] ->  ^H08    ;
        [ .C., 0 , ^H20 ,    1    ,    0    ] ->  ^H20    ;
        [ .C., 0 , ^H40 ,    1    ,    0    ] ->  ^H40    ;
        [ .C., 0 , ^HFF ,    1    ,    0    ] ->  ^HFF    ;
        [ .C., 0 , ^HFE ,    1    ,    0    ] ->  ^HFE    ;
        [ .C., 0 , ^HFB ,    1    ,    0    ] ->  ^HFB    ;
        [ .C., 0 , ^H7F ,    1    ,    0    ] ->  ^H7F    ;

        [ .C., 0 , ^HFF ,    1    ,    1    ] ->  ^H00    ;
```

```
[ .C., 0 , ^H01 ,    0    ,    0    ] ->  ^H01   ;
[ .C., 0 , ^H02 ,    0    ,    0    ] ->  ^H03   ;
[ .C., 0 , ^H04 ,    0    ,    0    ] ->  ^H07   ;
[ .C., 0 , ^H08 ,    0    ,    0    ] ->  ^H0F   ;
[ .C., 0 , ^H10 ,    0    ,    0    ] ->  ^H1F   ;
[ .C., 0 , ^H20 ,    0    ,    0    ] ->  ^H3F   ;
[ .C., 0 , ^H40 ,    0    ,    0    ] ->  ^H7F   ;
[ .C., 0 , ^H80 ,    0    ,    0    ] ->  ^HFF   ;
[ .C., 0 , ^HFE ,    0    ,    0    ] ->  ^H01   ;
[ .C., 0 , ^HFD ,    0    ,    0    ] ->  ^HFC   ;

[ .C., 0 , ^H5A ,    1    ,    0    ] ->  ^H5A   ;
[ .C., 0 , ^H87 ,    0    ,    1    ] ->  ^H5A   ;
[ .C., 0 , ^H5A ,    1    ,    0    ] ->  ^H5A   ;
[ .C., 0 , ^H87 ,    0    ,    1    ] ->  ^H5A   ;
[ .C., 0 , ^H0F ,    1    ,    0    ] ->  ^H0F   ;
[ .C., 0 , ^H1E ,    0    ,    1    ] ->  ^H0F   ;
[ .C., 0 , ^H0D ,    1    ,    0    ] ->  ^H0D   ;
[ .C., 0 , ^H1F ,    0    ,    1    ] ->  ^H0D   ;
[ .C., 0 , ^H0E ,    1    ,    0    ] ->  ^H0E   ;
[ .C., 0 , ^H1E ,    0    ,    0    ] ->  ^H10   ;
[ .C., 0 , ^HEE ,    1    ,    0    ] ->  ^HEE   ;
[ .C., 0 , ^HFE ,    0    ,    1    ] ->  ^HEE   ;

End XOR_Acc;
```

```
;************************************  ********************************;
;                                                                       ;
;               CHECKSUM / SYNDROME GENERATOR TABLES                    ;
;                                                                       ;
;       ( 4 byte checksum, double error correcting capability )         ;
;                                                                       ;
;                                                                       ;
;   Generator Polynomial :                                              ;
;                                                                       ;
;         0   4         76   3         251   2         81   1         10  ;
;  (alpha * x )+(alpha   * x )+(alpha    * x )+(alpha   * x )+(alpha   ) ;
;                                                                       ;
;                                                                       ;
;       10 September 1985       Craig Clapp                             ;
;       Copyright (C) PicTel Corporation, 1985                          ;
;                                                                       ;
;***********************************************************************;

                Name    CSGTable

Syndrome_3      segment byte at 0000H

; Linear to Linear Multiplication by Alpha ** 4

;                00      01      02      03      04      05      06      07

S3_00   db      000H , 010H , 020H , 030H , 040H , 050H , 060H , 070H
S3_08   db      080H , 090H , 0a0H , 0b0H , 0c0H , 0d0H , 0e0H , 0f0H
S3_10   db      01dH , 00dH , 03dH , 02dH , 05dH , 04dH , 07dH , 06dH
S3_18   db      09dH , 08dH , 0bdH , 0adH , 0ddH , 0cdH , 0fdH , 0edH
S3_20   db      03aH , 02aH , 01aH , 00aH , 07aH , 06aH , 05aH , 04aH
S3_28   db      0baH , 0aaH , 09aH , 08aH , 0faH , 0eaH , 0daH , 0caH
S3_30   db      027H , 037H , 007H , 017H , 067H , 077H , 047H , 057H
S3_38   db      0a7H , 0b7H , 087H , 097H , 0e7H , 0f7H , 0c7H , 0d7H
S3_40   db      074H , 064H , 054H , 044H , 034H , 024H , 014H , 004H
S3_48   db      0f4H , 0e4H , 0d4H , 0c4H , 0b4H , 0a4H , 094H , 084H
S3_50   db      069H , 079H , 049H , 059H , 029H , 039H , 009H , 019H
S3_58   db      0e9H , 0f9H , 0c9H , 0d9H , 0a9H , 0b9H , 089H , 099H
S3_60   db      04eH , 05eH , 06eH , 07eH , 00eH , 01eH , 02eH , 03eH
S3_68   db      0ceH , 0deH , 0eeH , 0feH , 08eH , 09eH , 0aeH , 0beH
S3_70   db      053H , 043H , 073H , 063H , 013H , 003H , 033H , 023H
S3_78   db      0d3H , 0c3H , 0f3H , 0e3H , 093H , 083H , 0b3H , 0a3H
S3_80   db      0e8H , 0f8H , 0c8H , 0d8H , 0a8H , 0b8H , 088H , 098H
S3_88   db      068H , 078H , 048H , 058H , 028H , 038H , 008H , 018H
S3_90   db      0f5H , 0e5H , 0d5H , 0c5H , 0b5H , 0a5H , 095H , 085H
S3_98   db      075H , 065H , 055H , 045H , 035H , 025H , 015H , 005H
S3_a0   db      0d2H , 0c2H , 0f2H , 0e2H , 092H , 082H , 0b2H , 0a2H
S3_a8   db      052H , 042H , 072H , 062H , 012H , 002H , 032H , 022H
S3_b0   db      0cfH , 0dfH , 0efH , 0ffH , 08fH , 09fH , 0afH , 0bfH
S3_b8   db      04fH , 05fH , 06fH , 07fH , 00fH , 01fH , 02fH , 03fH
S3_c0   db      09cH , 08cH , 0bcH , 0acH , 0dcH , 0ccH , 0fcH , 0ecH
S3_c8   db      01cH , 00cH , 03cH , 02cH , 05cH , 04cH , 07cH , 06cH
S3_d0   db      081H , 091H , 0a1H , 0b1H , 0c1H , 0d1H , 0e1H , 0f1H
S3_d8   db      001H , 011H , 021H , 031H , 041H , 051H , 061H , 071H
S3_e0   db      0a6H , 0b6H , 086H , 096H , 0e6H , 0f6H , 0c6H , 0d6H
S3_e8   db      026H , 036H , 006H , 016H , 066H , 076H , 046H , 056H
S3_f0   db      0bbH , 0abH , 09bH , 08bH , 0fbH , 0ebH , 0dbH , 0cbH
```

Attachment 6.
(7 p.)

```
S3_f8   db      03bH , 02bH , 01bH , 0   . , 07bH , 06bH , 05bH , 04bH

Syndrome_3      ends


Checkbyte_3     segment byte at 0010H

; Linear to Linear Multiplication by Alpha ** 10

;                00     01     02     03     04     05     06     07

C3_00   db      000H , 074H , 0e8H , 09cH , 0cdH , 0b9H , 025H , 051H
C3_08   db      087H , 0f3H , 06fH , 01bH , 04aH , 03eH , 0a2H , 0d6H
C3_10   db      013H , 067H , 0fbH , 08fH , 0deH , 0aaH , 036H , 042H
C3_18   db      094H , 0e0H , 07cH , 008H , 059H , 02dH , 0b1H , 0c5H
C3_20   db      026H , 052H , 0ceH , 0baH , 0ebH , 09fH , 003H , 077H
C3_28   db      0a1H , 0d5H , 049H , 03dH , 06cH , 018H , 084H , 0f0H
C3_30   db      035H , 041H , 0ddH , 0a9H , 0f8H , 08cH , 010H , 064H
C3_38   db      0b2H , 0c6H , 05aH , 02eH , 07fH , 00bH , 097H , 0e3H
C3_40   db      04cH , 038H , 0a4H , 0d0H , 081H , 0f5H , 069H , 01dH
C3_48   db      0cbH , 0bfH , 023H , 057H , 006H , 072H , 0eeH , 09aH
C3_50   db      05fH , 02bH , 0b7H , 0c3H , 092H , 0e6H , 07aH , 00eH
C3_58   db      0d8H , 0acH , 030H , 044H , 015H , 061H , 0fdH , 089H
C3_60   db      06aH , 01eH , 082H , 0f6H , 0a7H , 0d3H , 04fH , 03bH
C3_68   db      0edH , 099H , 005H , 071H , 020H , 054H , 0c8H , 0bcH
C3_70   db      079H , 00dH , 091H , 0e5H , 0b4H , 0c0H , 05cH , 028H
C3_78   db      0feH , 08aH , 016H , 062H , 033H , 047H , 0dbH , 0afH
C3_80   db      098H , 0ecH , 070H , 004H , 055H , 021H , 0bdH , 0c9H
C3_88   db      01fH , 06bH , 0f7H , 083H , 0d2H , 0a6H , 03aH , 04eH
C3_90   db      08bH , 0ffH , 063H , 017H , 046H , 032H , 0aeH , 0daH
C3_98   db      00cH , 078H , 0e4H , 090H , 0c1H , 0b5H , 029H , 05dH
C3_a0   db      0beH , 0caH , 056H , 022H , 073H , 007H , 09bH , 0efH
C3_a8   db      039H , 04dH , 0d1H , 0a5H , 0f4H , 080H , 01cH , 068H
C3_b0   db      0adH , 0d9H , 045H , 031H , 060H , 014H , 088H , 0fcH
C3_b8   db      02aH , 05eH , 0c2H , 0b6H , 0e7H , 093H , 00fH , 07bH
C3_c0   db      0d4H , 0a0H , 03cH , 048H , 019H , 06dH , 0f1H , 085H
C3_c8   db      053H , 027H , 0bbH , 0cfH , 09eH , 0eaH , 076H , 002H
C3_d0   db      0c7H , 0b3H , 02fH , 05bH , 00aH , 07eH , 0e2H , 096H
C3_d8   db      040H , 034H , 0a8H , 0dcH , 08dH , 0f9H , 065H , 011H
C3_e0   db      0f2H , 086H , 01aH , 06eH , 03fH , 04bH , 0d7H , 0a3H
C3_e8   db      075H , 001H , 09dH , 0e9H , 0b8H , 0ccH , 050H , 024H
C3_f0   db      0e1H , 095H , 009H , 07dH , 02cH , 058H , 0c4H , 0b0H
C3_f8   db      066H , 012H , 08eH , 0faH , 0abH , 0dfH , 043H , 037H

Checkbyte_3     ends


Syndrome_2      segment byte at 0020H

; Linear to Linear Multiplication by Alpha ** 3

;                00     01     02     03     04     05     06     07

S2_00   db      000H , 008H , 010H , 018H , 020H , 028H , 030H , 038H
S2_08   db      040H , 048H , 050H , 058H , 060H , 068H , 070H , 078H
S2_10   db      080H , 088H , 090H , 098H , 0a0H , 0a8H , 0b0H , 0b8H
S2_18   db      0c0H , 0c8H , 0d0H , 0d8H , 0e0H , 0e8H , 0f0H , 0f8H
S2_20   db      01dH , 015H , 00dH , 005H , 03dH , 035H , 02dH , 025H
```

```
S2_28   db      05dH , 055H , 04dH , 0   , 07dH , 075H , 06dH , 065H
S2_30   db      09dH , 095H , 08dH , 085H , 0bdH , 0b5H , 0adH , 0a5H
S2_38   db      0ddH , 0d5H , 0cdH , 0c5H , 0fdH , 0f5H , 0edH , 0e5H
S2_40   db      03aH , 032H , 02aH , 022H , 01aH , 012H , 00aH , 002H
S2_48   db      07aH , 072H , 06aH , 062H , 05aH , 052H , 04aH , 042H
S2_50   db      0baH , 0b2H , 0aaH , 0a2H , 09aH , 092H , 08aH , 082H
S2_58   db      0faH , 0f2H , 0eaH , 0e2H , 0daH , 0d2H , 0caH , 0c2H
S2_60   db      027H , 02fH , 037H , 03fH , 007H , 00fH , 017H , 01fH
S2_68   db      067H , 06fH , 077H , 07fH , 047H , 04fH , 057H , 05fH
S2_70   db      0a7H , 0afH , 0b7H , 0bfH , 087H , 08fH , 097H , 09fH
S2_78   db      0e7H , 0efH , 0f7H , 0ffH , 0c7H , 0cfH , 0d7H , 0dfH
S2_80   db      074H , 07cH , 064H , 06cH , 054H , 05cH , 044H , 04cH
S2_88   db      034H , 03cH , 024H , 02cH , 014H , 01cH , 004H , 00cH
S2_90   db      0f4H , 0fcH , 0e4H , 0ecH , 0d4H , 0dcH , 0c4H , 0ccH
S2_98   db      0b4H , 0bcH , 0a4H , 0acH , 094H , 09cH , 084H , 08cH
S2_a0   db      069H , 061H , 079H , 071H , 049H , 041H , 059H , 051H
S2_a8   db      029H , 021H , 039H , 031H , 009H , 001H , 019H , 011H
S2_b0   db      0e9H , 0e1H , 0f9H , 0f1H , 0c9H , 0c1H , 0d9H , 0d1H
S2_b8   db      0a9H , 0a1H , 0b9H , 0b1H , 089H , 081H , 099H , 091H
S2_c0   db      04eH , 046H , 05eH , 056H , 06eH , 066H , 07eH , 076H
S2_c8   db      00eH , 006H , 01eH , 016H , 02eH , 026H , 03eH , 036H
S2_d0   db      0ceH , 0c6H , 0deH , 0d6H , 0eeH , 0e6H , 0feH , 0f6H
S2_d8   db      08eH , 086H , 09eH , 096H , 0aeH , 0a6H , 0beH , 0b6H
S2_e0   db      053H , 05bH , 043H , 04bH , 073H , 07bH , 063H , 06bH
S2_e8   db      013H , 01bH , 003H , 00bH , 033H , 03bH , 023H , 02bH
S2_f0   db      0d3H , 0dbH , 0c3H , 0cbH , 0f3H , 0fbH , 0e3H , 0ebH
S2_f8   db      093H , 09bH , 083H , 08bH , 0b3H , 0bbH , 0a3H , 0abH

Syndrome_2      ends


Checkbyte_2     segment byte at 0030H

; Linear to Linear Multiplication by Alpha ** 81

;                00      01      02      03      04      05      06      07

C2_00   db      000H , 0e7H , 0d3H , 034H , 0bbH , 05cH , 068H , 08fH
C2_08   db      06bH , 08cH , 0b8H , 05fH , 0d0H , 037H , 003H , 0e4H
C2_10   db      0d6H , 031H , 005H , 0e2H , 06dH , 08aH , 0beH , 059H
C2_18   db      0bdH , 05aH , 06eH , 089H , 006H , 0e1H , 0d5H , 032H
C2_20   db      0b1H , 056H , 062H , 085H , 00aH , 0edH , 0d9H , 03eH
C2_28   db      0daH , 03dH , 009H , 0eeH , 061H , 086H , 0b2H , 055H
C2_30   db      067H , 080H , 0b4H , 053H , 0dcH , 03bH , 00fH , 0e8H
C2_38   db      00cH , 0ebH , 0dfH , 038H , 0b7H , 050H , 064H , 083H
C2_40   db      07fH , 098H , 0acH , 04bH , 0c4H , 023H , 017H , 0f0H
C2_48   db      014H , 0f3H , 0c7H , 020H , 0afH , 048H , 07cH , 09bH
C2_50   db      0a9H , 04eH , 07aH , 09dH , 012H , 0f5H , 0c1H , 026H
C2_58   db      0c2H , 025H , 011H , 0f6H , 079H , 09eH , 0aaH , 04dH
C2_60   db      0ceH , 029H , 01dH , 0faH , 075H , 092H , 0a6H , 041H
C2_68   db      0a5H , 042H , 076H , 091H , 01eH , 0f9H , 0cdH , 02aH
C2_70   db      018H , 0ffH , 0cbH , 02cH , 0a3H , 044H , 070H , 097H
C2_78   db      073H , 094H , 0a0H , 047H , 0c8H , 02fH , 01bH , 0fcH
C2_80   db      0feH , 019H , 02dH , 0caH , 045H , 0a2H , 096H , 071H
C2_88   db      095H , 072H , 046H , 0a1H , 02eH , 0c9H , 0fdH , 01aH
C2_90   db      028H , 0cfH , 0fbH , 01cH , 093H , 074H , 040H , 0a7H
C2_98   db      043H , 0a4H , 090H , 077H , 0f8H , 01fH , 02bH , 0ccH
C2_a0   db      04fH , 0a8H , 09cH , 07bH , 0f4H , 013H , 027H , 0c0H
```

```
C2_a8   db      024H , 0c3H , 0f7H , 0[illegible] , 09fH , 078H , 04cH , 0abH
C2_b0   db      099H , 07eH , 04aH , 0adH , 022H , 0c5H , 0f1H , 016H
C2_b8   db      0f2H , 015H , 021H , 0c6H , 049H , 0aeH , 09aH , 07dH
C2_c0   db      081H , 066H , 052H , 0b5H , 03aH , 0ddH , 0e9H , 00eH
C2_c8   db      0eaH , 00dH , 039H , 0deH , 051H , 0b6H , 082H , 065H
C2_d0   db      057H , 0b0H , 084H , 063H , 0ecH , 00bH , 03fH , 0d8H
C2_d8   db      03cH , 0dbH , 0efH , 008H , 087H , 060H , 054H , 0b3H
C2_e0   db      030H , 0d7H , 0e3H , 004H , 08bH , 06cH , 058H , 0bfH
C2_e8   db      05bH , 0bcH , 088H , 06fH , 0e0H , 007H , 033H , 0d4H
C2_f0   db      0e6H , 001H , 035H , 0d2H , 05dH , 0baH , 08eH , 069H
C2_f8   db      08dH , 06aH , 05eH , 0b9H , 036H , 0d1H , 0e5H , 002H

Checkbyte_2     ends


Syndrome_1      segment byte at 0040H

; Linear to Linear Multiplication by Alpha ** 2

;                00      01      02      03      04      05      06      07

S1_00   db      000H , 004H , 008H , 00cH , 010H , 014H , 018H , 01cH
S1_08   db      020H , 024H , 028H , 02cH , 030H , 034H , 038H , 03cH
S1_10   db      040H , 044H , 048H , 04cH , 050H , 054H , 058H , 05cH
S1_18   db      060H , 064H , 068H , 06cH , 070H , 074H , 078H , 07cH
S1_20   db      080H , 084H , 088H , 08cH , 090H , 094H , 098H , 09cH
S1_28   db      0a0H , 0a4H , 0a8H , 0acH , 0b0H , 0b4H , 0b8H , 0bcH
S1_30   db      0c0H , 0c4H , 0c8H , 0ccH , 0d0H , 0d4H , 0d8H , 0dcH
S1_38   db      0e0H , 0e4H , 0e8H , 0ecH , 0f0H , 0f4H , 0f8H , 0fcH
S1_40   db      01dH , 019H , 015H , 011H , 00dH , 009H , 005H , 001H
S1_48   db      03dH , 039H , 035H , 031H , 02dH , 029H , 025H , 021H
S1_50   db      05dH , 059H , 055H , 051H , 04dH , 049H , 045H , 041H
S1_58   db      07dH , 079H , 075H , 071H , 06dH , 069H , 065H , 061H
S1_60   db      09dH , 099H , 095H , 091H , 08dH , 089H , 085H , 081H
S1_68   db      0bdH , 0b9H , 0b5H , 0b1H , 0adH , 0a9H , 0a5H , 0a1H
S1_70   db      0ddH , 0d9H , 0d5H , 0d1H , 0cdH , 0c9H , 0c5H , 0c1H
S1_78   db      0fdH , 0f9H , 0f5H , 0f1H , 0edH , 0e9H , 0e5H , 0e1H
S1_80   db      03aH , 03eH , 032H , 036H , 02aH , 02eH , 022H , 026H
S1_88   db      01aH , 01eH , 012H , 016H , 00aH , 00eH , 002H , 006H
S1_90   db      07aH , 07eH , 072H , 076H , 06aH , 06eH , 062H , 066H
S1_98   db      05aH , 05eH , 052H , 056H , 04aH , 04eH , 042H , 046H
S1_a0   db      0baH , 0beH , 0b2H , 0b6H , 0aaH , 0aeH , 0a2H , 0a6H
S1_a8   db      09aH , 09eH , 092H , 096H , 08aH , 08eH , 082H , 086H
S1_b0   db      0faH , 0feH , 0f2H , 0f6H , 0eaH , 0eeH , 0e2H , 0e6H
S1_b8   db      0daH , 0deH , 0d2H , 0d6H , 0caH , 0ceH , 0c2H , 0c6H
S1_c0   db      027H , 023H , 02fH , 02bH , 037H , 033H , 03fH , 03bH
S1_c8   db      007H , 003H , 00fH , 00bH , 017H , 013H , 01fH , 01bH
S1_d0   db      067H , 063H , 06fH , 06bH , 077H , 073H , 07fH , 07bH
S1_d8   db      047H , 043H , 04fH , 04bH , 057H , 053H , 05fH , 05bH
S1_e0   db      0a7H , 0a3H , 0afH , 0abH , 0b7H , 0b3H , 0bfH , 0bbH
S1_e8   db      087H , 083H , 08fH , 08bH , 097H , 093H , 09fH , 09bH
S1_f0   db      0e7H , 0e3H , 0efH , 0ebH , 0f7H , 0f3H , 0ffH , 0fbH
S1_f8   db      0c7H , 0c3H , 0cfH , 0cbH , 0d7H , 0d3H , 0dfH , 0dbH

Syndrome_1      ends


Checkbyte_1     segment byte at 0050H
```

```
; Linear to Linear Multiplication by Alpha ** 251

;                00      01      02      03      04      05      06      07

C1_00   db      000H , 0d8H , 0adH , 075H , 047H , 09fH , 0eaH , 032H
C1_08   db      08eH , 056H , 023H , 0fbH , 0c9H , 011H , 064H , 0bcH
C1_10   db      001H , 0d9H , 0acH , 074H , 046H , 09eH , 0ebH , 033H
C1_18   db      08fH , 057H , 022H , 0faH , 0c8H , 010H , 065H , 0bdH
C1_20   db      002H , 0daH , 0afH , 077H , 045H , 09dH , 0e8H , 030H
C1_28   db      08cH , 054H , 021H , 0f9H , 0cbH , 013H , 066H , 0beH
C1_30   db      003H , 0dbH , 0aeH , 076H , 044H , 09cH , 0e9H , 031H
C1_38   db      08dH , 055H , 020H , 0f8H , 0caH , 012H , 067H , 0bfH
C1_40   db      004H , 0dcH , 0a9H , 071H , 043H , 09bH , 0eeH , 036H
C1_48   db      08aH , 052H , 027H , 0ffH , 0cdH , 015H , 060H , 0b8H
C1_50   db      005H , 0ddH , 0a8H , 070H , 042H , 09aH , 0efH , 037H
C1_58   db      08bH , 053H , 026H , 0feH , 0ccH , 014H , 061H , 0b9H
C1_60   db      006H , 0deH , 0abH , 073H , 041H , 099H , 0ecH , 034H
C1_68   db      088H , 050H , 025H , 0fdH , 0cfH , 017H , 062H , 0baH
C1_70   db      007H , 0dfH , 0aaH , 072H , 040H , 098H , 0edH , 035H
C1_78   db      089H , 051H , 024H , 0fcH , 0ceH , 016H , 063H , 0bbH
C1_80   db      008H , 0d0H , 0a5H , 07dH , 04fH , 097H , 0e2H , 03aH
C1_88   db      086H , 05eH , 02bH , 0f3H , 0c1H , 019H , 06cH , 0b4H
C1_90   db      009H , 0d1H , 0a4H , 07cH , 04eH , 096H , 0e3H , 03bH
C1_98   db      087H , 05fH , 02aH , 0f2H , 0c0H , 018H , 06dH , 0b5H
C1_a0   db      00aH , 0d2H , 0a7H , 07fH , 04dH , 095H , 0e0H , 038H
C1_a8   db      084H , 05cH , 029H , 0f1H , 0c3H , 01bH , 06eH , 0b6H
C1_b0   db      00bH , 0d3H , 0a6H , 07eH , 04cH , 094H , 0e1H , 039H
C1_b8   db      085H , 05dH , 028H , 0f0H , 0c2H , 01aH , 06fH , 0b7H
C1_c0   db      00cH , 0d4H , 0a1H , 079H , 04bH , 093H , 0e6H , 03eH
C1_c8   db      082H , 05aH , 02fH , 0f7H , 0c5H , 01dH , 068H , 0b0H
C1_d0   db      00dH , 0d5H , 0a0H , 078H , 04aH , 092H , 0e7H , 03fH
C1_d8   db      083H , 05bH , 02eH , 0f6H , 0c4H , 01cH , 069H , 0b1H
C1_e0   db      00eH , 0d6H , 0a3H , 07bH , 049H , 091H , 0e4H , 03cH
C1_e8   db      080H , 058H , 02dH , 0f5H , 0c7H , 01fH , 06aH , 0b2H
C1_f0   db      00fH , 0d7H , 0a2H , 07aH , 048H , 090H , 0e5H , 03dH
C1_f8   db      081H , 059H , 02cH , 0f4H , 0c6H , 01eH , 06bH , 0b3H

Checkbyte_1     ends


Syndrome_0      segment byte at 0060H

; Linear to Linear Multiplication by Alpha ** 1

;                00      01      02      03      04      05      06      07

S0_00   db      000H , 002H , 004H , 006H , 008H , 00aH , 00cH , 00eH
S0_08   db      010H , 012H , 014H , 016H , 018H , 01aH , 01cH , 01eH
S0_10   db      020H , 022H , 024H , 026H , 028H , 02aH , 02cH , 02eH
S0_18   db      030H , 032H , 034H , 036H , 038H , 03aH , 03cH , 03eH
S0_20   db      040H , 042H , 044H , 046H , 048H , 04aH , 04cH , 04eH
S0_28   db      050H , 052H , 054H , 056H , 058H , 05aH , 05cH , 05eH
S0_30   db      060H , 062H , 064H , 066H , 068H , 06aH , 06cH , 06eH
S0_38   db      070H , 072H , 074H , 076H , 078H , 07aH , 07cH , 07eH
S0_40   db      080H , 082H , 084H , 086H , 088H , 08aH , 08cH , 08eH
S0_48   db      090H , 092H , 094H , 096H , 098H , 09aH , 09cH , 09eH
S0_50   db      0a0H , 0a2H , 0a4H , 0a6H , 0a8H , 0aaH , 0acH , 0aeH
```

```
S0_58   db      0b0H , 0b2H , 0b4H ,  H , 0b8H , 0baH , 0bcH , 0beH
S0_60   db      0c0H , 0c2H , 0c4H , 0c6H , 0c8H , 0caH , 0ccH , 0ceH
S0_68   db      0d0H , 0d2H , 0d4H , 0d6H , 0d8H , 0daH , 0dcH , 0deH
S0_70   db      0e0H , 0e2H , 0e4H , 0e6H , 0e8H , 0eaH , 0ecH , 0eeH
S0_78   db      0f0H , 0f2H , 0f4H , 0f6H , 0f8H , 0faH , 0fcH , 0feH
S0_80   db      01dH , 01fH , 019H , 01bH , 015H , 017H , 011H , 013H
S0_88   db      00dH , 00fH , 009H , 00bH , 005H , 007H , 001H , 003H
S0_90   db      03dH , 03fH , 039H , 03bH , 035H , 037H , 031H , 033H
S0_98   db      02dH , 02fH , 029H , 02bH , 025H , 027H , 021H , 023H
S0_a0   db      05dH , 05fH , 059H , 05bH , 055H , 057H , 051H , 053H
S0_a8   db      04dH , 04fH , 049H , 04bH , 045H , 047H , 041H , 043H
S0_b0   db      07dH , 07fH , 079H , 07bH , 075H , 077H , 071H , 073H
S0_b8   db      06dH , 06fH , 069H , 06bH , 065H , 067H , 061H , 063H
S0_c0   db      09dH , 09fH , 099H , 09bH , 095H , 097H , 091H , 093H
S0_c8   db      08dH , 08fH , 089H , 08bH , 085H , 087H , 081H , 083H
S0_d0   db      0bdH , 0bfH , 0b9H , 0bbH , 0b5H , 0b7H , 0b1H , 0b3H
S0_d8   db      0adH , 0afH , 0a9H , 0abH , 0a5H , 0a7H , 0a1H , 0a3H
S0_e0   db      0ddH , 0dfH , 0d9H , 0dbH , 0d5H , 0d7H , 0d1H , 0d3H
S0_e8   db      0cdH , 0cfH , 0c9H , 0cbH , 0c5H , 0c7H , 0c1H , 0c3H
S0_f0   db      0fdH , 0ffH , 0f9H , 0fbH , 0f5H , 0f7H , 0f1H , 0f3H
S0_f8   db      0edH , 0efH , 0e9H , 0ebH , 0e5H , 0e7H , 0e1H , 0e3H

Syndrome_0      ends


Checkbyte_0     segment byte at 0070H

; Linear to Linear Multiplication by Alpha ** 76

;                00     01     02     03     04     05     06     07

C0_00   db      000H , 01eH , 03cH , 022H , 078H , 066H , 044H , 05aH
C0_08   db      0f0H , 0eeH , 0ccH , 0d2H , 088H , 096H , 0b4H , 0aaH
C0_10   db      0fdH , 0e3H , 0c1H , 0dfH , 085H , 09bH , 0b9H , 0a7H
C0_18   db      00dH , 013H , 031H , 02fH , 075H , 06bH , 049H , 057H
C0_20   db      0e7H , 0f9H , 0dbH , 0c5H , 09fH , 081H , 0a3H , 0bdH
C0_28   db      017H , 009H , 02bH , 035H , 06fH , 071H , 053H , 04dH
C0_30   db      01aH , 004H , 026H , 038H , 062H , 07cH , 05eH , 040H
C0_38   db      0eaH , 0f4H , 0d6H , 0c8H , 092H , 08cH , 0aeH , 0b0H
C0_40   db      0d3H , 0cdH , 0efH , 0f1H , 0abH , 0b5H , 097H , 089H
C0_48   db      023H , 03dH , 01fH , 001H , 05bH , 045H , 067H , 079H
C0_50   db      02eH , 030H , 012H , 00cH , 056H , 048H , 06aH , 074H
C0_58   db      0deH , 0c0H , 0e2H , 0fcH , 0a6H , 0b8H , 09aH , 084H
C0_60   db      034H , 02aH , 008H , 016H , 04cH , 052H , 070H , 06eH
C0_68   db      0c4H , 0daH , 0f8H , 0e6H , 0bcH , 0a2H , 080H , 09eH
C0_70   db      0c9H , 0d7H , 0f5H , 0ebH , 0b1H , 0afH , 08dH , 093H
C0_78   db      039H , 027H , 005H , 01bH , 041H , 05fH , 07dH , 063H
C0_80   db      0bbH , 0a5H , 087H , 099H , 0c3H , 0ddH , 0ffH , 0e1H
C0_88   db      04bH , 055H , 077H , 069H , 033H , 02dH , 00fH , 011H
C0_90   db      046H , 058H , 07aH , 064H , 03eH , 020H , 002H , 01cH
C0_98   db      0b6H , 0a8H , 08aH , 094H , 0ceH , 0d0H , 0f2H , 0ecH
C0_a0   db      05cH , 042H , 060H , 07eH , 024H , 03aH , 018H , 006H
C0_a8   db      0acH , 0b2H , 090H , 08eH , 0d4H , 0caH , 0e8H , 0f6H
C0_b0   db      0a1H , 0bfH , 09dH , 083H , 0d9H , 0c7H , 0e5H , 0fbH
C0_b8   db      051H , 04fH , 06dH , 073H , 029H , 037H , 015H , 00bH
C0_c0   db      068H , 076H , 054H , 04aH , 010H , 00eH , 02cH , 032H
C0_c8   db      098H , 086H , 0a4H , 0baH , 0e0H , 0feH , 0dcH , 0c2H
C0_d0   db      095H , 08bH , 0a9H , 0b7H , 0edH , 0f3H , 0d1H , 0cfH
```

```
C0_d8   db      065H , 07bH , 059H , 0[illegible]1 , 01dH , 003H , 021H , 03fH
C0_e0   db      08fH , 091H , 0b3H , 0adH , 0f7H , 0e9H , 0cbH , 0d5H
C0_e8   db      07fH , 061H , 043H , 05dH , 007H , 019H , 03bH , 025H
C0_f0   db      072H , 06cH , 04eH , 050H , 00aH , 014H , 036H , 028H
C0_f8   db      082H , 09cH , 0beH , 0a0H , 0faH , 0e4H , 0c6H , 0d8H

Checkbyte_0     ends

        end
```

```
/************************************  ********************************
 *                                                                      *
 *      sio_errs - Receive error correction procedure.                  *
 *                                                                      *
 *      Abstract                                                        *
 *      --------                                                        *
 *                                                                      *
 *      sio_errs implements a procedure to correct up to two bytes      *
 *      in error in a received packet.                                  *
 *                                                                      *
 *      History                                                         *
 *      -------                                                         *
 *                                                                      *
 *      07/01/85        Bernard Szabo   Original version.               *
 *      06/09/86        Paul Alexander  Revisited.                      *
 *      02/08/88        Bill Kruger     Incorporated for Gemini         *
 *                                      Just changed name and added     *
 *                                      LUT load routine                *
 *                                                                      *
 ************************************************************************/

#ifdef vax11c

#include "machine"      /* machine dependent defines and typedefs       */
#include "error"        /* system error codes                           */
#include "gcp"          /* gcp board addressing and contants            */
#include "siointf"      /* Serial I/O Driver Public Interface           */
#include "sio"          /* Serial I/O Driver Private Data Definitions   */

#else

#include "machine.h"    /* machine dependent defines and typedefs       */
#include "error.h"      /* system error codes                           */
#include "gcp.h"        /* gcp board addressing and contants            */
#include "siointf.h"    /* Serial I/O Driver Public Interface           */
#include "sio.h"        /* Serial I/O Driver Private Data Definitions   */

#endif
```

Attachment 7.
(20 p.)

```
/************************************   *********************************
 *      Constants associated with GF(256) and with actual block length.
 ***************************************************************************/

#define alpha_linear_zero 255   /* alpha representation of linear zero  */
#define size_of_gf        255   /* size of multiplic. group in GF(256)  */
#define max_alpha_in_gf   254   /* maximum alpha representation in group*/

                                /* max # significant symbols in block   */
#define max_block_len     SIO_RPKT_LEN


/***************************************************************************
 *      sio_error - Debug variables associated with sio_r_correct
 ***************************************************************************/

globaldef USHORT sio_error[2][2];
```

```
/************************************  ********************************
 *      Hardware Lookup Tables                                        *
 ************************************************************************/

                                /* These tables are needed by the error */
                                /* correction hardware in order to      */
                                /* generate checksums and syndromes.    */
                                /* They are loaded into predefined      */
                                /* locations in the CI buffer memory at */
                                /* system initialization time           */

                                /* linear to linear mult by alpha**76   */
                                /* loaded as checksum table 4           */
static UCHAR alpha_76[256] =
{
    0x000, 0x01e, 0x03c, 0x022, 0x078, 0x066, 0x044, 0x05a,
    0x0f0, 0x0ee, 0x0cc, 0x0d2, 0x088, 0x096, 0x0b4, 0x0aa,
    0x0fd, 0x0e3, 0x0c1, 0x0df, 0x085, 0x09b, 0x0b9, 0x0a7,
    0x00d, 0x013, 0x031, 0x02f, 0x075, 0x06b, 0x049, 0x057,
    0x0e7, 0x0f9, 0x0db, 0x0c5, 0x09f, 0x081, 0x0a3, 0x0bd,
    0x017, 0x009, 0x02b, 0x035, 0x06f, 0x071, 0x053, 0x04d,
    0x01a, 0x004, 0x026, 0x038, 0x062, 0x07c, 0x05e, 0x040,
    0x0ea, 0x0f4, 0x0d6, 0x0c8, 0x092, 0x08c, 0x0ae, 0x0b0,
    0x0d3, 0x0cd, 0x0ef, 0x0f1, 0x0ab, 0x0b5, 0x097, 0x089,
    0x023, 0x03d, 0x01f, 0x001, 0x05b, 0x045, 0x067, 0x079,
    0x02e, 0x030, 0x012, 0x00c, 0x056, 0x048, 0x06a, 0x074,
    0x0de, 0x0c0, 0x0e2, 0x0fc, 0x0a6, 0x0b8, 0x09a, 0x084,
    0x034, 0x02a, 0x008, 0x016, 0x04c, 0x052, 0x070, 0x06e,
    0x0c4, 0x0da, 0x0f8, 0x0e6, 0x0bc, 0x0a2, 0x080, 0x09e,
    0x0c9, 0x0d7, 0x0f5, 0x0eb, 0x0b1, 0x0af, 0x08d, 0x093,
    0x039, 0x027, 0x005, 0x01b, 0x041, 0x05f, 0x07d, 0x063,
    0x0bb, 0x0a5, 0x087, 0x099, 0x0c3, 0x0dd, 0x0ff, 0x0e1,
    0x04b, 0x055, 0x077, 0x069, 0x033, 0x02d, 0x00f, 0x011,
    0x046, 0x058, 0x07a, 0x064, 0x03e, 0x020, 0x002, 0x01c,
    0x0b6, 0x0a8, 0x08a, 0x094, 0x0ce, 0x0d0, 0x0f2, 0x0ec,
    0x05c, 0x042, 0x060, 0x07e, 0x024, 0x03a, 0x018, 0x006,
    0x0ac, 0x0b2, 0x090, 0x08e, 0x0d4, 0x0ca, 0x0e8, 0x0f6,
    0x0a1, 0x0bf, 0x09d, 0x083, 0x0d9, 0x0c7, 0x0e5, 0x0fb,
    0x051, 0x04f, 0x06d, 0x073, 0x029, 0x037, 0x015, 0x00b,
    0x068, 0x076, 0x054, 0x04a, 0x010, 0x00e, 0x02c, 0x032,
    0x098, 0x086, 0x0a4, 0x0ba, 0x0e0, 0x0fe, 0x0dc, 0x0c2,
    0x095, 0x08b, 0x0a9, 0x0b7, 0x0ed, 0x0f3, 0x0d1, 0x0cf,
    0x065, 0x07b, 0x059, 0x047, 0x01d, 0x003, 0x021, 0x03f,
    0x08f, 0x091, 0x0b3, 0x0ad, 0x0f7, 0x0e9, 0x0cb, 0x0d5,
    0x07f, 0x061, 0x043, 0x05d, 0x007, 0x019, 0x03b, 0x025,
    0x072, 0x06c, 0x04e, 0x050, 0x00a, 0x014, 0x036, 0x028,
    0x082, 0x09c, 0x0be, 0x0a0, 0x0fa, 0x0e4, 0x0c6, 0x0d8
};

                                /* linear to linear mult by alpha**251  */
                                /* loaded as checksum table 3           */
static UCHAR alpha_251[256] =
{
    0x000, 0x0d8, 0x0ad, 0x075, 0x047, 0x09f, 0x0ea, 0x032,
    0x08e, 0x056, 0x023, 0x0fb, 0x0c9, 0x011, 0x064, 0x0bc,
    0x001, 0x0d9, 0x0ac, 0x074, 0x046, 0x09e, 0x0eb, 0x033,
    0x08f, 0x057, 0x022, 0x0fa, 0x0c8, 0x010, 0x065, 0x0bd,
    0x002, 0x0da, 0x0af, 0x077, 0x045, 0x09d, 0x0e8, 0x030,
```

```
    0x08c, 0x054, 0x021, 0x0f9, 0x0cb,  .013, 0x066, 0x0be,
    0x003, 0x0db, 0x0ae, 0x076, 0x044, 0x09c, 0x0e9, 0x031,
    0x08d, 0x055, 0x020, 0x0f8, 0x0ca, 0x012, 0x067, 0x0bf,
    0x004, 0x0dc, 0x0a9, 0x071, 0x043, 0x09b, 0x0ee, 0x036,
    0x08a, 0x052, 0x027, 0x0ff, 0x0cd, 0x015, 0x060, 0x0b8,
    0x005, 0x0dd, 0x0a8, 0x070, 0x042, 0x09a, 0x0ef, 0x037,
    0x08b, 0x053, 0x026, 0x0fe, 0x0cc, 0x014, 0x061, 0x0b9,
    0x006, 0x0de, 0x0ab, 0x073, 0x041, 0x099, 0x0ec, 0x034,
    0x088, 0x050, 0x025, 0x0fd, 0x0cf, 0x017, 0x062, 0x0ba,
    0x007, 0x0df, 0x0aa, 0x072, 0x040, 0x098, 0x0ed, 0x035,
    0x089, 0x051, 0x024, 0x0fc, 0x0ce, 0x016, 0x063, 0x0bb,
    0x008, 0x0d0, 0x0a5, 0x07d, 0x04f, 0x097, 0x0e2, 0x03a,
    0x086, 0x05e, 0x02b, 0x0f3, 0x0c1, 0x019, 0x06c, 0x0b4,
    0x009, 0x0d1, 0x0a4, 0x07c, 0x04e, 0x096, 0x0e3, 0x03b,
    0x087, 0x05f, 0x02a, 0x0f2, 0x0c0, 0x018, 0x06d, 0x0b5,
    0x00a, 0x0d2, 0x0a7, 0x07f, 0x04d, 0x095, 0x0e0, 0x038,
    0x084, 0x05c, 0x029, 0x0f1, 0x0c3, 0x01b, 0x06e, 0x0b6,
    0x00b, 0x0d3, 0x0a6, 0x07e, 0x04c, 0x094, 0x0e1, 0x039,
    0x085, 0x05d, 0x028, 0x0f0, 0x0c2, 0x01a, 0x06f, 0x0b7,
    0x00c, 0x0d4, 0x0a1, 0x079, 0x04b, 0x093, 0x0e6, 0x03e,
    0x082, 0x05a, 0x02f, 0x0f7, 0x0c5, 0x01d, 0x068, 0x0b0,
    0x00d, 0x0d5, 0x0a0, 0x078, 0x04a, 0x092, 0x0e7, 0x03f,
    0x083, 0x05b, 0x02e, 0x0f6, 0x0c4, 0x01c, 0x069, 0x0b1,
    0x00e, 0x0d6, 0x0a3, 0x07b, 0x049, 0x091, 0x0e4, 0x03c,
    0x080, 0x058, 0x02d, 0x0f5, 0x0c7, 0x01f, 0x06a, 0x0b2,
    0x00f, 0x0d7, 0x0a2, 0x07a, 0x048, 0x090, 0x0e5, 0x03d,
    0x081, 0x059, 0x02c, 0x0f4, 0x0c6, 0x01e, 0x06b, 0x0b3
};

                                /* linear to linear mult by alpha**81    */
                                /* loaded as checksum table 2            */
static UCHAR alpha_81[256] =
{
    0x000, 0x0e7, 0x0d3, 0x034, 0x0bb, 0x05c, 0x068, 0x08f,
    0x06b, 0x08c, 0x0b8, 0x05f, 0x0d0, 0x037, 0x003, 0x0e4,
    0x0d6, 0x031, 0x005, 0x0e2, 0x06d, 0x08a, 0x0be, 0x059,
    0x0bd, 0x05a, 0x06e, 0x089, 0x006, 0x0e1, 0x0d5, 0x032,
    0x0b1, 0x056, 0x062, 0x085, 0x00a, 0x0ed, 0x0d9, 0x03e,
    0x0da, 0x03d, 0x009, 0x0ee, 0x061, 0x086, 0x0b2, 0x055,
    0x067, 0x080, 0x0b4, 0x053, 0x0dc, 0x03b, 0x00f, 0x0e8,
    0x00c, 0x0eb, 0x0df, 0x038, 0x0b7, 0x050, 0x064, 0x083,
    0x07f, 0x098, 0x0ac, 0x04b, 0x0c4, 0x023, 0x017, 0x0f0,
    0x014, 0x0f3, 0x0c7, 0x020, 0x0af, 0x048, 0x07c, 0x09b,
    0x0a9, 0x04e, 0x07a, 0x09d, 0x012, 0x0f5, 0x0c1, 0x026,
    0x0c2, 0x025, 0x011, 0x0f6, 0x079, 0x09e, 0x0aa, 0x04d,
    0x0ce, 0x029, 0x01d, 0x0fa, 0x075, 0x092, 0x0a6, 0x041,
    0x0a5, 0x042, 0x076, 0x091, 0x01e, 0x0f9, 0x0cd, 0x02a,
    0x018, 0x0ff, 0x0cb, 0x02c, 0x0a3, 0x044, 0x070, 0x097,
    0x073, 0x094, 0x0a0, 0x047, 0x0c8, 0x02f, 0x01b, 0x0fc,
    0x0fe, 0x019, 0x02d, 0x0ca, 0x045, 0x0a2, 0x096, 0x071,
    0x095, 0x072, 0x046, 0x0a1, 0x02e, 0x0c9, 0x0fd, 0x01a,
    0x028, 0x0cf, 0x0fb, 0x01c, 0x093, 0x074, 0x040, 0x0a7,
    0x043, 0x0a4, 0x090, 0x077, 0x0f8, 0x01f, 0x02b, 0x0cc,
    0x04f, 0x0a8, 0x09c, 0x07b, 0x0f4, 0x013, 0x027, 0x0c0,
    0x024, 0x0c3, 0x0f7, 0x010, 0x09f, 0x078, 0x04c, 0x0ab,
    0x099, 0x07e, 0x04a, 0x0ad, 0x022, 0x0c5, 0x0f1, 0x016,
    0x0f2, 0x015, 0x021, 0x0c6, 0x049, 0x0ae, 0x09a, 0x07d,
    0x081, 0x066, 0x052, 0x0b5, 0x03a, 0x0dd, 0x0e9, 0x00e,
```

```
    0x0ea, 0x00d, 0x039, 0x0de, 0x051, 0x0b6, 0x082, 0x065,
    0x057, 0x0b0, 0x084, 0x063, 0x0ec, 0x00b, 0x03f, 0x0d8,
    0x03c, 0x0db, 0x0ef, 0x008, 0x087, 0x060, 0x054, 0x0b3,
    0x030, 0x0d7, 0x0e3, 0x004, 0x08b, 0x06c, 0x058, 0x0bf,
    0x05b, 0x0bc, 0x088, 0x06f, 0x0e0, 0x007, 0x033, 0x0d4,
    0x0e6, 0x001, 0x035, 0x0d2, 0x05d, 0x0ba, 0x08e, 0x069,
    0x08d, 0x06a, 0x05e, 0x0b9, 0x036, 0x0d1, 0x0e5, 0x002
};

                                /* linear to linear mult by alpha**10   */
                                /* loaded as checksum table 1           */
static UCHAR alpha_10[256] =
{
    0x000, 0x074, 0x0e8, 0x09c, 0x0cd, 0x0b9, 0x025, 0x051,
    0x087, 0x0f3, 0x06f, 0x01b, 0x04a, 0x03e, 0x0a2, 0x0d6,
    0x013, 0x067, 0x0fb, 0x08f, 0x0de, 0x0aa, 0x036, 0x042,
    0x094, 0x0e0, 0x07c, 0x008, 0x059, 0x02d, 0x0b1, 0x0c5,
    0x026, 0x052, 0x0ce, 0x0ba, 0x0eb, 0x09f, 0x003, 0x077,
    0x0a1, 0x0d5, 0x049, 0x03d, 0x06c, 0x018, 0x084, 0x0f0,
    0x035, 0x041, 0x0dd, 0x0a9, 0x0f8, 0x08c, 0x010, 0x064,
    0x0b2, 0x0c6, 0x05a, 0x02e, 0x07f, 0x00b, 0x097, 0x0e3,
    0x04c, 0x038, 0x0a4, 0x0d0, 0x081, 0x0f5, 0x069, 0x01d,
    0x0cb, 0x0bf, 0x023, 0x057, 0x006, 0x072, 0x0ee, 0x09a,
    0x05f, 0x02b, 0x0b7, 0x0c3, 0x092, 0x0e6, 0x07a, 0x00e,
    0x0d8, 0x0ac, 0x030, 0x044, 0x015, 0x061, 0x0fd, 0x089,
    0x06a, 0x01e, 0x082, 0x0f6, 0x0a7, 0x0d3, 0x04f, 0x03b,
    0x0ed, 0x099, 0x005, 0x071, 0x020, 0x054, 0x0c8, 0x0bc,
    0x079, 0x00d, 0x091, 0x0e5, 0x0b4, 0x0c0, 0x05c, 0x028,
    0x0fe, 0x08a, 0x016, 0x062, 0x033, 0x047, 0x0db, 0x0af,
    0x098, 0x0ec, 0x070, 0x004, 0x055, 0x021, 0x0bd, 0x0c9,
    0x01f, 0x06b, 0x0f7, 0x083, 0x0d2, 0x0a6, 0x03a, 0x04e,
    0x08b, 0x0ff, 0x063, 0x017, 0x046, 0x032, 0x0ae, 0x0da,
    0x00c, 0x078, 0x0e4, 0x090, 0x0c1, 0x0b5, 0x029, 0x05d,
    0x0be, 0x0ca, 0x056, 0x022, 0x073, 0x007, 0x09b, 0x0ef,
    0x039, 0x04d, 0x0d1, 0x0a5, 0x0f4, 0x080, 0x01c, 0x068,
    0x0ad, 0x0d9, 0x045, 0x031, 0x060, 0x014, 0x088, 0x0fc,
    0x02a, 0x05e, 0x0c2, 0x0b6, 0x0e7, 0x093, 0x00f, 0x07b,
    0x0d4, 0x0a0, 0x03c, 0x048, 0x019, 0x06d, 0x0f1, 0x085,
    0x053, 0x027, 0x0bb, 0x0cf, 0x09e, 0x0ea, 0x076, 0x002,
    0x0c7, 0x0b3, 0x02f, 0x05b, 0x00a, 0x07e, 0x0e2, 0x096,
    0x040, 0x034, 0x0a8, 0x0dc, 0x08d, 0x0f9, 0x065, 0x011,
    0x0f2, 0x086, 0x01a, 0x06e, 0x03f, 0x04b, 0x0d7, 0x0a3,
    0x075, 0x001, 0x09d, 0x0e9, 0x0b8, 0x0cc, 0x050, 0x024,
    0x0e1, 0x095, 0x009, 0x07d, 0x02c, 0x058, 0x0c4, 0x0b0,
    0x066, 0x012, 0x08e, 0x0fa, 0x0ab, 0x0df, 0x043, 0x037,
};

                                /* linear to linear mult by alpha**1    */
                                /* loaded as syndrome table 1           */
static UCHAR alpha_1[256] =
{
    0x000, 0x002, 0x004, 0x006, 0x008, 0x00a, 0x00c, 0x00e,
    0x010, 0x012, 0x014, 0x016, 0x018, 0x01a, 0x01c, 0x01e,
    0x020, 0x022, 0x024, 0x026, 0x028, 0x02a, 0x02c, 0x02e,
    0x030, 0x032, 0x034, 0x036, 0x038, 0x03a, 0x03c, 0x03e,
    0x040, 0x042, 0x044, 0x046, 0x048, 0x04a, 0x04c, 0x04e,
    0x050, 0x052, 0x054, 0x056, 0x058, 0x05a, 0x05c, 0x05e,
    0x060, 0x062, 0x064, 0x066, 0x068, 0x06a, 0x06c, 0x06e,
```

```
    0x070, 0x072, 0x074, 0x076, 0x078,  x07a, 0x07c, 0x07e,
    0x080, 0x082, 0x084, 0x086, 0x088, 0x08a, 0x08c, 0x08e,
    0x090, 0x092, 0x094, 0x096, 0x098, 0x09a, 0x09c, 0x09e,
    0x0a0, 0x0a2, 0x0a4, 0x0a6, 0x0a8, 0x0aa, 0x0ac, 0x0ae,
    0x0b0, 0x0b2, 0x0b4, 0x0b6, 0x0b8, 0x0ba, 0x0bc, 0x0be,
    0x0c0, 0x0c2, 0x0c4, 0x0c6, 0x0c8, 0x0ca, 0x0cc, 0x0ce,
    0x0d0, 0x0d2, 0x0d4, 0x0d6, 0x0d8, 0x0da, 0x0dc, 0x0de,
    0x0e0, 0x0e2, 0x0e4, 0x0e6, 0x0e8, 0x0ea, 0x0ec, 0x0ee,
    0x0f0, 0x0f2, 0x0f4, 0x0f6, 0x0f8, 0x0fa, 0x0fc, 0x0fe,
    0x01d, 0x01f, 0x019, 0x01b, 0x015, 0x017, 0x011, 0x013,
    0x00d, 0x00f, 0x009, 0x00b, 0x005, 0x007, 0x001, 0x003,
    0x03d, 0x03f, 0x039, 0x03b, 0x035, 0x037, 0x031, 0x033,
    0x02d, 0x02f, 0x029, 0x02b, 0x025, 0x027, 0x021, 0x023,
    0x05d, 0x05f, 0x059, 0x05b, 0x055, 0x057, 0x051, 0x053,
    0x04d, 0x04f, 0x049, 0x04b, 0x045, 0x047, 0x041, 0x043,
    0x07d, 0x07f, 0x079, 0x07b, 0x075, 0x077, 0x071, 0x073,
    0x06d, 0x06f, 0x069, 0x06b, 0x065, 0x067, 0x061, 0x063,
    0x09d, 0x09f, 0x099, 0x09b, 0x095, 0x097, 0x091, 0x093,
    0x08d, 0x08f, 0x089, 0x08b, 0x085, 0x087, 0x081, 0x083,
    0x0bd, 0x0bf, 0x0b9, 0x0bb, 0x0b5, 0x0b7, 0x0b1, 0x0b3,
    0x0ad, 0x0af, 0x0a9, 0x0ab, 0x0a5, 0x0a7, 0x0a1, 0x0a3,
    0x0dd, 0x0df, 0x0d9, 0x0db, 0x0d5, 0x0d7, 0x0d1, 0x0d3,
    0x0cd, 0x0cf, 0x0c9, 0x0cb, 0x0c5, 0x0c7, 0x0c1, 0x0c3,
    0x0fd, 0x0ff, 0x0f9, 0x0fb, 0x0f5, 0x0f7, 0x0f1, 0x0f3,
    0x0ed, 0x0ef, 0x0e9, 0x0eb, 0x0e5, 0x0e7, 0x0e1, 0x0e3
};

                                /* linear to linear mult by alpha**2    */
                                /* loaded as syndrome table 2           */
static UCHAR alpha_2[256] =
{
    0x000, 0x004, 0x008, 0x00c, 0x010, 0x014, 0x018, 0x01c,
    0x020, 0x024, 0x028, 0x02c, 0x030, 0x034, 0x038, 0x03c,
    0x040, 0x044, 0x048, 0x04c, 0x050, 0x054, 0x058, 0x05c,
    0x060, 0x064, 0x068, 0x06c, 0x070, 0x074, 0x078, 0x07c,
    0x080, 0x084, 0x088, 0x08c, 0x090, 0x094, 0x098, 0x09c,
    0x0a0, 0x0a4, 0x0a8, 0x0ac, 0x0b0, 0x0b4, 0x0b8, 0x0bc,
    0x0c0, 0x0c4, 0x0c8, 0x0cc, 0x0d0, 0x0d4, 0x0d8, 0x0dc,
    0x0e0, 0x0e4, 0x0e8, 0x0ec, 0x0f0, 0x0f4, 0x0f8, 0x0fc,
    0x01d, 0x019, 0x015, 0x011, 0x00d, 0x009, 0x005, 0x001,
    0x03d, 0x039, 0x035, 0x031, 0x02d, 0x029, 0x025, 0x021,
    0x05d, 0x059, 0x055, 0x051, 0x04d, 0x049, 0x045, 0x041,
    0x07d, 0x079, 0x075, 0x071, 0x06d, 0x069, 0x065, 0x061,
    0x09d, 0x099, 0x095, 0x091, 0x08d, 0x089, 0x085, 0x081,
    0x0bd, 0x0b9, 0x0b5, 0x0b1, 0x0ad, 0x0a9, 0x0a5, 0x0a1,
    0x0dd, 0x0d9, 0x0d5, 0x0d1, 0x0cd, 0x0c9, 0x0c5, 0x0c1,
    0x0fd, 0x0f9, 0x0f5, 0x0f1, 0x0ed, 0x0e9, 0x0e5, 0x0e1,
    0x03a, 0x03e, 0x032, 0x036, 0x02a, 0x02e, 0x022, 0x026,
    0x01a, 0x01e, 0x012, 0x016, 0x00a, 0x00e, 0x002, 0x006,
    0x07a, 0x07e, 0x072, 0x076, 0x06a, 0x06e, 0x062, 0x066,
    0x05a, 0x05e, 0x052, 0x056, 0x04a, 0x04e, 0x042, 0x046,
    0x0ba, 0x0be, 0x0b2, 0x0b6, 0x0aa, 0x0ae, 0x0a2, 0x0a6,
    0x09a, 0x09e, 0x092, 0x096, 0x08a, 0x08e, 0x082, 0x086,
    0x0fa, 0x0fe, 0x0f2, 0x0f6, 0x0ea, 0x0ee, 0x0e2, 0x0e6,
    0x0da, 0x0de, 0x0d2, 0x0d6, 0x0ca, 0x0ce, 0x0c2, 0x0c6,
    0x027, 0x023, 0x02f, 0x02b, 0x037, 0x033, 0x03f, 0x03b,
    0x007, 0x003, 0x00f, 0x00b, 0x017, 0x013, 0x01f, 0x01b,
    0x067, 0x063, 0x06f, 0x06b, 0x077, 0x073, 0x07f, 0x07b,
```

```
    0x047, 0x043, 0x04f, 0x04b, 0x057, 0x053, 0x05f, 0x05b,
    0x0a7, 0x0a3, 0x0af, 0x0ab, 0x0b7, 0x0b3, 0x0bf, 0x0bb,
    0x087, 0x083, 0x08f, 0x08b, 0x097, 0x093, 0x09f, 0x09b,
    0x0e7, 0x0e3, 0x0ef, 0x0eb, 0x0f7, 0x0f3, 0x0ff, 0x0fb,
    0x0c7, 0x0c3, 0x0cf, 0x0cb, 0x0d7, 0x0d3, 0x0df, 0x0db
};

                                /* linear to linear mult by alpha**3    */
                                /* loaded as syndrome table 3           */
static UCHAR alpha_3[256] =
{
    0x000, 0x008, 0x010, 0x018, 0x020, 0x028, 0x030, 0x038,
    0x040, 0x048, 0x050, 0x058, 0x060, 0x068, 0x070, 0x078,
    0x080, 0x088, 0x090, 0x098, 0x0a0, 0x0a8, 0x0b0, 0x0b8,
    0x0c0, 0x0c8, 0x0d0, 0x0d8, 0x0e0, 0x0e8, 0x0f0, 0x0f8,
    0x01d, 0x015, 0x00d, 0x005, 0x03d, 0x035, 0x02d, 0x025,
    0x05d, 0x055, 0x04d, 0x045, 0x07d, 0x075, 0x06d, 0x065,
    0x09d, 0x095, 0x08d, 0x085, 0x0bd, 0x0b5, 0x0ad, 0x0a5,
    0x0dd, 0x0d5, 0x0cd, 0x0c5, 0x0fd, 0x0f5, 0x0ed, 0x0e5,
    0x03a, 0x032, 0x02a, 0x022, 0x01a, 0x012, 0x00a, 0x002,
    0x07a, 0x072, 0x06a, 0x062, 0x05a, 0x052, 0x04a, 0x042,
    0x0ba, 0x0b2, 0x0aa, 0x0a2, 0x09a, 0x092, 0x08a, 0x082,
    0x0fa, 0x0f2, 0x0ea, 0x0e2, 0x0da, 0x0d2, 0x0ca, 0x0c2,
    0x027, 0x02f, 0x037, 0x03f, 0x007, 0x00f, 0x017, 0x01f,
    0x067, 0x06f, 0x077, 0x07f, 0x047, 0x04f, 0x057, 0x05f,
    0x0a7, 0x0af, 0x0b7, 0x0bf, 0x087, 0x08f, 0x097, 0x09f,
    0x0e7, 0x0ef, 0x0f7, 0x0ff, 0x0c7, 0x0cf, 0x0d7, 0x0df,
    0x074, 0x07c, 0x064, 0x06c, 0x054, 0x05c, 0x044, 0x04c,
    0x034, 0x03c, 0x024, 0x02c, 0x014, 0x01c, 0x004, 0x00c,
    0x0f4, 0x0fc, 0x0e4, 0x0ec, 0x0d4, 0x0dc, 0x0c4, 0x0cc,
    0x0b4, 0x0bc, 0x0a4, 0x0ac, 0x094, 0x09c, 0x084, 0x08c,
    0x069, 0x061, 0x079, 0x071, 0x049, 0x041, 0x059, 0x051,
    0x029, 0x021, 0x039, 0x031, 0x009, 0x001, 0x019, 0x011,
    0x0e9, 0x0e1, 0x0f9, 0x0f1, 0x0c9, 0x0c1, 0x0d9, 0x0d1,
    0x0a9, 0x0a1, 0x0b9, 0x0b1, 0x089, 0x081, 0x099, 0x091,
    0x04e, 0x046, 0x05e, 0x056, 0x06e, 0x066, 0x07e, 0x076,
    0x00e, 0x006, 0x01e, 0x016, 0x02e, 0x026, 0x03e, 0x036,
    0x0ce, 0x0c6, 0x0de, 0x0d6, 0x0ee, 0x0e6, 0x0fe, 0x0f6,
    0x08e, 0x086, 0x09e, 0x096, 0x0ae, 0x0a6, 0x0be, 0x0b6,
    0x053, 0x05b, 0x043, 0x04b, 0x073, 0x07b, 0x063, 0x06b,
    0x013, 0x01b, 0x003, 0x00b, 0x033, 0x03b, 0x023, 0x02b,
    0x0d3, 0x0db, 0x0c3, 0x0cb, 0x0f3, 0x0fb, 0x0e3, 0x0eb,
    0x093, 0x09b, 0x083, 0x08b, 0x0b3, 0x0bb, 0x0a3, 0x0ab
};

                                /* linear to linear mult by alpha**4    */
                                /* loaded as syndrome table 4           */
static UCHAR alpha_4[256] =
{
    0x000, 0x010, 0x020, 0x030, 0x040, 0x050, 0x060, 0x070,
    0x080, 0x090, 0x0a0, 0x0b0, 0x0c0, 0x0d0, 0x0e0, 0x0f0,
    0x01d, 0x00d, 0x03d, 0x02d, 0x05d, 0x04d, 0x07d, 0x06d,
    0x09d, 0x08d, 0x0bd, 0x0ad, 0x0dd, 0x0cd, 0x0fd, 0x0ed,
    0x03a, 0x02a, 0x01a, 0x00a, 0x07a, 0x06a, 0x05a, 0x04a,
    0x0ba, 0x0aa, 0x09a, 0x08a, 0x0fa, 0x0ea, 0x0da, 0x0ca,
    0x027, 0x037, 0x007, 0x017, 0x067, 0x077, 0x047, 0x057,
    0x0a7, 0x0b7, 0x087, 0x097, 0x0e7, 0x0f7, 0x0c7, 0x0d7,
    0x074, 0x064, 0x054, 0x044, 0x034, 0x024, 0x014, 0x004,
```

```
    0x0f4, 0x0e4, 0x0d4, 0x0c4, 0x0b4,  .0a4, 0x094, 0x084,
    0x069, 0x079, 0x049, 0x059, 0x029, 0x039, 0x009, 0x019,
    0x0e9, 0x0f9, 0x0c9, 0x0d9, 0x0a9, 0x0b9, 0x089, 0x099,
    0x04e, 0x05e, 0x06e, 0x07e, 0x00e, 0x01e, 0x02e, 0x03e,
    0x0ce, 0x0de, 0x0ee, 0x0fe, 0x08e, 0x09e, 0x0ae, 0x0be,
    0x053, 0x043, 0x073, 0x063, 0x013, 0x003, 0x033, 0x023,
    0x0d3, 0x0c3, 0x0f3, 0x0e3, 0x093, 0x083, 0x0b3, 0x0a3,
    0x0e8, 0x0f8, 0x0c8, 0x0d8, 0x0a8, 0x0b8, 0x088, 0x098,
    0x068, 0x078, 0x048, 0x058, 0x028, 0x038, 0x008, 0x018,
    0x0f5, 0x0e5, 0x0d5, 0x0c5, 0x0b5, 0x0a5, 0x095, 0x085,
    0x075, 0x065, 0x055, 0x045, 0x035, 0x025, 0x015, 0x005,
    0x0d2, 0x0c2, 0x0f2, 0x0e2, 0x092, 0x082, 0x0b2, 0x0a2,
    0x052, 0x042, 0x072, 0x062, 0x012, 0x002, 0x032, 0x022,
    0x0cf, 0x0df, 0x0ef, 0x0ff, 0x08f, 0x09f, 0x0af, 0x0bf,
    0x04f, 0x05f, 0x06f, 0x07f, 0x00f, 0x01f, 0x02f, 0x03f,
    0x09c, 0x08c, 0x0bc, 0x0ac, 0x0dc, 0x0cc, 0x0fc, 0x0ec,
    0x01c, 0x00c, 0x03c, 0x02c, 0x05c, 0x04c, 0x07c, 0x06c,
    0x081, 0x091, 0x0a1, 0x0b1, 0x0c1, 0x0d1, 0x0e1, 0x0f1,
    0x001, 0x011, 0x021, 0x031, 0x041, 0x051, 0x061, 0x071,
    0x0a6, 0x0b6, 0x086, 0x096, 0x0e6, 0x0f6, 0x0c6, 0x0d6,
    0x026, 0x036, 0x006, 0x016, 0x066, 0x076, 0x046, 0x056,
    0x0bb, 0x0ab, 0x09b, 0x08b, 0x0fb, 0x0eb, 0x0db, 0x0cb,
    0x03b, 0x02b, 0x01b, 0x00b, 0x07b, 0x06b, 0x05b, 0x04b
};
```

```
/************************************  ********************************
 *      lin_to_log - Table to convert from linear to alpha representation.
 *
 *      description:    This table maps elements from the linear
 *                      (polynomial) representation of GF(256) to
 *                      the alpha (logarithmic) representation.
 ************************************************************************/

static UCHAR lin_to_log[size_of_gf+1] =
{
        0377, 0000, 0001, 0031, 0002, 0062, 0032, 0306,
        0003, 0337, 0063, 0356, 0033, 0150, 0307, 0113,
        0004, 0144, 0340, 0016, 0064, 0215, 0357, 0201,
        0034, 0301, 0151, 0370, 0310, 0010, 0114, 0161,
        0005, 0212, 0145, 0057, 0341, 0044, 0017, 0041,
        0065, 0223, 0216, 0332, 0360, 0022, 0202, 0105,
        0035, 0265, 0302, 0175, 0152, 0047, 0371, 0271,
        0311, 0232, 0011, 0170, 0115, 0344, 0162, 0246,
        0006, 0277, 0213, 0142, 0146, 0335, 0060, 0375,
        0342, 0230, 0045, 0263, 0020, 0221, 0042, 0210,
        0066, 0320, 0224, 0316, 0217, 0226, 0333, 0275,
        0361, 0322, 0023, 0134, 0203, 0070, 0106, 0100,
        0036, 0102, 0266, 0243, 0303, 0110, 0176, 0156,
        0153, 0072, 0050, 0124, 0372, 0205, 0272, 0075,
        0312, 0136, 0233, 0237, 0012, 0025, 0171, 0053,
        0116, 0324, 0345, 0254, 0163, 0363, 0247, 0127,
        0007, 0160, 0300, 0367, 0214, 0200, 0143, 0015,
        0147, 0112, 0336, 0355, 0061, 0305, 0376, 0030,
        0343, 0245, 0231, 0167, 0046, 0270, 0264, 0174,
        0021, 0104, 0222, 0331, 0043, 0040, 0211, 0056,
        0067, 0077, 0321, 0133, 0225, 0274, 0317, 0315,
        0220, 0207, 0227, 0262, 0334, 0374, 0276, 0141,
        0362, 0126, 0323, 0253, 0024, 0052, 0135, 0236,
        0204, 0074, 0071, 0123, 0107, 0155, 0101, 0242,
        0037, 0055, 0103, 0330, 0267, 0173, 0244, 0166,
        0304, 0027, 0111, 0354, 0177, 0014, 0157, 0366,
        0154, 0241, 0073, 0122, 0051, 0235, 0125, 0252,
        0373, 0140, 0206, 0261, 0273, 0314, 0076, 0132,
        0313, 0131, 0137, 0260, 0234, 0251, 0240, 0121,
        0013, 0365, 0026, 0353, 0172, 0165, 0054, 0327,
        0117, 0256, 0325, 0351, 0346, 0347, 0255, 0350,
        0164, 0326, 0364, 0352, 0250, 0120, 0130, 0257
};
```

```
/*************************************************************************
 *      log_to_lin - Table to convert from alpha to linear representation.
 *
 *      description:    This table maps elements from the alpha
 *                      (logarithmic) representation of GF(256) to
 *                      the linear (polynomial) representation.
 *************************************************************************/

static UCHAR log_to_lin[size_of_gf+1] =
{
        0001, 0002, 0004, 0010, 0020, 0040, 0100, 0200,
        0035, 0072, 0164, 0350, 0315, 0207, 0023, 0046,
        0114, 0230, 0055, 0132, 0264, 0165, 0352, 0311,
        0217, 0003, 0006, 0014, 0030, 0060, 0140, 0300,
        0235, 0047, 0116, 0234, 0045, 0112, 0224, 0065,
        0152, 0324, 0265, 0167, 0356, 0301, 0237, 0043,
        0106, 0214, 0005, 0012, 0024, 0050, 0120, 0240,
        0135, 0272, 0151, 0322, 0271, 0157, 0336, 0241,
        0137, 0276, 0141, 0302, 0231, 0057, 0136, 0274,
        0145, 0312, 0211, 0017, 0036, 0074, 0170, 0360,
        0375, 0347, 0323, 0273, 0153, 0326, 0261, 0177,
        0376, 0341, 0337, 0243, 0133, 0266, 0161, 0342,
        0331, 0257, 0103, 0206, 0021, 0042, 0104, 0210,
        0015, 0032, 0064, 0150, 0320, 0275, 0147, 0316,
        0201, 0037, 0076, 0174, 0370, 0355, 0307, 0223,
        0073, 0166, 0354, 0305, 0227, 0063, 0146, 0314,
        0205, 0027, 0056, 0134, 0270, 0155, 0332, 0251,
        0117, 0236, 0041, 0102, 0204, 0025, 0052, 0124,
        0250, 0115, 0232, 0051, 0122, 0244, 0125, 0252,
        0111, 0222, 0071, 0162, 0344, 0325, 0267, 0163,
        0346, 0321, 0277, 0143, 0306, 0221, 0077, 0176,
        0374, 0345, 0327, 0263, 0173, 0366, 0361, 0377,
        0343, 0333, 0253, 0113, 0226, 0061, 0142, 0304,
        0225, 0067, 0156, 0334, 0245, 0127, 0256, 0101,
        0202, 0031, 0062, 0144, 0310, 0215, 0007, 0016,
        0034, 0070, 0160, 0340, 0335, 0247, 0123, 0246,
        0121, 0242, 0131, 0262, 0171, 0362, 0371, 0357,
        0303, 0233, 0053, 0126, 0254, 0105, 0212, 0011,
        0022, 0044, 0110, 0220, 0075, 0172, 0364, 0365,
        0367, 0363, 0373, 0353, 0313, 0213, 0013, 0026,
        0054, 0130, 0260, 0175, 0372, 0351, 0317, 0203,
        0033, 0066, 0154, 0330, 0255, 0107, 0216, 0000
};
```

```
/************************************  ********************************
 *      soln_table - Table used to find root of error locator polynomial.
 *
 *      description:    This table is used to solve the error locator
 *                      polynomial: y**2 + y + c = 0.
 *
 *                      The table maps c => y_2. The two roots y_1,y_2
 *                      are related by: y_1 = y_2 + a**0.
 *
 *                      Note that c,y_1,y_2 are in alpha representation.
 ************************************************************************/

static UCHAR soln_table[size_of_gf+1] =
{
        0252, 0365, 0353, 0360, 0327, 0377, 0341, 0320,
        0257, 0377, 0377, 0377, 0303, 0214, 0241, 0377,
        0260, 0231, 0377, 0307, 0377, 0377, 0377, 0376,
        0220, 0246, 0031, 0244, 0330, 0377, 0377, 0025,
        0276, 0377, 0356, 0324, 0377, 0313, 0226, 0377,
        0377, 0312, 0377, 0377, 0377, 0321, 0375, 0377,
        0041, 0264, 0344, 0267, 0062, 0377, 0354, 0377,
        0261, 0377, 0377, 0237, 0377, 0377, 0052, 0377,
        0302, 0377, 0377, 0043, 0335, 0377, 0251, 0377,
        0377, 0362, 0262, 0326, 0055, 0377, 0377, 0377,
        0377, 0377, 0274, 0377, 0377, 0104, 0377, 0377,
        0377, 0377, 0266, 0377, 0373, 0377, 0377, 0377,
        0102, 0377, 0370, 0316, 0311, 0377, 0366, 0363,
        0144, 0332, 0377, 0377, 0331, 0377, 0377, 0134,
        0143, 0357, 0377, 0377, 0377, 0377, 0077, 0160,
        0377, 0377, 0377, 0342, 0124, 0377, 0377, 0323,
        0372, 0170, 0377, 0150, 0377, 0377, 0106, 0377,
        0314, 0343, 0377, 0177, 0123, 0122, 0377, 0212,
        0377, 0152, 0345, 0377, 0145, 0377, 0350, 0377,
        0132, 0333, 0377, 0377, 0377, 0317, 0377, 0377,
        0377, 0221, 0377, 0377, 0171, 0153, 0377, 0377,
        0377, 0377, 0210, 0377, 0377, 0377, 0377, 0377,
        0377, 0147, 0377, 0371, 0155, 0377, 0377, 0211,
        0367, 0377, 0377, 0203, 0377, 0161, 0377, 0351,
        0204, 0215, 0377, 0377, 0361, 0277, 0235, 0202,
        0223, 0377, 0377, 0377, 0355, 0377, 0347, 0377,
        0310, 0377, 0265, 0377, 0377, 0377, 0377, 0377,
        0263, 0374, 0377, 0304, 0377, 0301, 0270, 0364,
        0306, 0377, 0337, 0242, 0377, 0377, 0377, 0377,
        0377, 0377, 0377, 0377, 0176, 0213, 0340, 0172,
        0377, 0240, 0377, 0377, 0377, 0377, 0305, 0272,
        0250, 0377, 0377, 0236, 0377, 0256, 0247, 0377
};
```

```
/************************************  ********************************
 *                                                                      *
 *      sio_err_init - Initialize Error Correction Subsystem            *
 *                                                                      *
 * Description : This routine does any initialization necessary for     *
 *               the error correction subsystem.  This consists of :    *
 *                                                                      *
 *      1. Load the checksum and syndrome lookup tables into CI memory  *
 *                                                                      *
 * Return Status : RET_OK               -       Successful init         *
 *                 ER_cimem_xx          -       CI mem alloc problem    *
 *                                                                      *
 ************************************************************************/

sio_err_init()
{
    UCHAR *tbl_addr;            /* destination address of each LUT      */
    USHORT i;                   /* loop iterand                         */
    USHORT status;              /* return status from CI mem alloc      */
    UCHAR *luts[8];             /* load order array of LUT addresses    */


                                /* allocate CI memory for LUTs          */
                                /* NOTE: These tables MUST be located   */
                                /* at the beginning of the CI memory.   */
                                /* Therefore, no CI mem allocs should   */
                                /* be done before this one!!!           */
    if ((status = ci_mem_alloc(&tbl_addr,(ULONG) 8 * 256)) != RET_OK)
                                /* CI mem alloc problem - abort         */
        return(status);
                                /* set up LUT pointer array             */

                                        /* checksum generator tables    */
    luts[0] = alpha_10;
    luts[1] = alpha_81;
    luts[2] = alpha_251;
    luts[3] = alpha_76;
                                        /* syndrome generator tables    */
    luts[4] = alpha_1;
    luts[5] = alpha_2;
    luts[6] = alpha_3;
    luts[7] = alpha_4;
                                /* load each table                      */
    for (i = 0; i < 8; i++)
    {
                                /* load the table                       */
        move_block_b(tbl_addr, luts[i], 256);
                                /* move table address to next tbl addr  */
        tbl_addr += 256;
    }
                                /* return successful init code          */
    return(RET_OK);
}
```

```
/*************************************. ***********************************
 *      alpha_add - addition operation in alpha representation of GF(256).
 *
 *      syntax:
 *
 *          sum = alpha_add(arg1, arg2);
 *
 *          arg1, arg2 - addends in alpha representation.
 *          sum -        sum in alpha representation.
 *
 *      description:
 *
 *          Convert arg1 & arg2 to polynomial representation.
 *          Perform the addition, which consists of taking
 *          the exclusive or. Finally, convert the sum back
 *          to alpha representation.
 ***************************************************************************/

USHORT alpha_add(arg1, arg2)

USHORT arg1, arg2;              /* Addends in alpha representation      */
{
    USHORT sum;                 /* Sum in alpha representation          */

                                /* Convert to polynomial representation */
                                /* and calculate the sum                */
    sum = log_to_lin[arg1] ^ log_to_lin[arg2];
                                /* Convert back to alpha and return     */
    return(lin_to_log[sum]);
}
```

```
/************************************  ********************************
 *      alpha_mul - multiplication in alpha representation of GF(256).
 *
 *      syntax:
 *
 *          product = alpha_mul(arg1, arg2);
 *
 *          arg1, arg2 - multiplicands in alpha representation.
 *          product -    product in alpha representation.
 *
 *      description:
 *
 *          If either multiplicand is the alpha representation
 *          of zero, return the same. Otherwise, perform the
 *          multiplication, which consists of taking the sum
 *          modulo 255.
 ************************************************************************/

USHORT alpha_mul(arg1, arg2)

USHORT arg1, arg2;              /* multiplicands in alpha representation*/
{
    USHORT prod;                /* product in alpha representation      */

                                /* case where one factor is zero        */
    if (arg1 == alpha_linear_zero || arg2 == alpha_linear_zero)
        return(alpha_linear_zero);
                                /* add exponents and return value       */
    prod = arg1 + arg2;
    return((prod > max_alpha_in_gf) ? prod - size_of_gf : prod);
}
```

```
/************************************  ********************************
 *      alpha_inv - multiplicative inverse in alpha represent. of GF(256).
 *
 *      syntax:
 *
 *          inverse = alpha_inv(arg);
 *
 *          arg -     term to invert in alpha representation.
 *          inverse - inverse in alpha representation.
 *
 *      description:
 *
 *          If the argument is the alpha representation of
 *          zero, return the same. Otherwise, return its
 *          additive inverse modulo 255.
 ************************************************************************/

USHORT alpha_inv(arg)

USHORT arg;                     /* argument to invert in alpha          */
{
                                /* return inverse of argument           */
    return((arg == alpha_linear_zero || arg == 0) ? arg : (size_of_gf - arg));
}
```

```
/************************************  ********************************
 *      solve_poly - find root of error locator polynomial
 *
 *      syntax:
 *
 *          root = solve_poly(sig_1, sig_2);
 *
 *          sig_1 - 1st degree coeff. of polynomial
 *          sig_2 - 0th degree coeff. of polynomial
 *          root -  Root of error locator polynomial
 *
 *      description:
 *
 *          Transform the error locator polynomial given by:
 *
 *              x**2 + sig_1*x + sig_2 = 0
 *
 *          via the substitutions:   x = sig_1*y,
 *                                   c = sig_2 / (sig_1**2)
 *
 *          to the polynomial:  y**2 + y + c = 0.
 *
 *          Find a root to this equation via table lookup
 *          and convert this to a root of the original.
 *
 *          Note that we are working within the alpha
 *          representation of GF(256).
 ************************************************************************/

USHORT solve_poly(sig_1, sig_2)

USHORT sig_1, sig_2;            /* Coefficients of polynomial           */
{
    USHORT ratio;               /* The derived coefficient c            */
    USHORT y_2;                 /* Root of derived polynomial           */

                                /* c = sig_2 / (sig_1**2)               */
    ratio = alpha_mul(sig_2, alpha_inv(alpha_mul(sig_1, sig_1)));
                                /* find y_2 given c via table           */
    y_2 = soln_table[ratio];
                                /* Return root of original equ          */
                                /* Which is: x_2 = sig_1 * y_2          */
    return(alpha_mul(sig_1, y_2));
}
```

```
/************************************   *********************************
 *      sio_r_correct - perform error correction on packet
 *
 *      syntax:
 *
 *          number = sio_r_correct(received, syndrome);
 *
 *          received -  pointer to received block
 *                      (pointer to last recv'd octet)
 *          syndrome -  pointer to array of syndromes
 *          number -    # corrected symbols. >2 indicates that the errors
 *                      could not be corrected.
 *
 *      description:
 *
 *          Perform error correction based on the computed
 *          syndromes. Return an indication of the number
 *          of symbols which were found in error.
 *
 *          The algorithm used for error correction is found in:
 *
 *          Practical Error Correction Design for Engineers,
 *          Neal Glover,
 *          Data Systems Technology.
 *
 *          The octets of the received packet, which consists
 *          of the packet control, data and checksum fields,
 *          are interpreted as the low order coefficients of
 *          a polynomial of degree 254 over the field GF(256).
 *          The actual packet length is less than 255, the high
 *          order coefficients are set to zero. High order co-
 *          efficients are transmitted and received first. Thus
 *          the highest order non-trivial term corresponds to
 *          the control field, the zero order term to the final
 *          octet of the checksum.
 *
 *          The polynomial (linear) representation is used for
 *          coding the message and calculating the syndromes.
 *          Note that the error correction algorithm uses the
 *          alpha (logarithmic) representation.
 *
 *          The syndromes are related to the error contributions
 *          and locations as follows:
 *
 *          s_i = e_1 * a**(i * l_1) + e_2 * a**(i * l_2)
 *
 *          where:  s_1 .. s_4 are the syndromes in alpha format,
 *                  l_1, l_2 represent the degrees in error,
 *                  e_1, e_2 represent the errors in alpha format.
 ************************************************************************/

USHORT sio_r_correct(received, syndrome)

UCHAR *received;                /* pointer to received block            */
UCHAR *syndrome;                /* pointer to array of syndromes        */
{
    USHORT  l_1, l_2;           /* degrees of coefficients in error     */
    USHORT  e_1, e_2;           /* error contributions in alpha         */
```

```
    USHORT  s_1, s_2, s_3, s_4; /* syn  ome values in alpha format      */
    USHORT  sig_1, sig_2;       /* coeff of error locator polynomial    */

/*      If the received syndromes are all zeroes, then no symbols       */
/*      are in error. Check for this case first.                        */

    if ((syndrome[0] | syndrome[1] | syndrome[2] | syndrome[3]) == 0)

        return(0);

/*      Otherwise, convert the syndromes to linear format               */

    s_1 = lin_to_log[syndrome[0]];
    s_2 = lin_to_log[syndrome[1]];
    s_3 = lin_to_log[syndrome[2]];
    s_4 = lin_to_log[syndrome[3]];

/*      If we have any of s_4/s_3, s_3/s_2, s_2/s_1 equal, then         */
/*      they are all equal and one symbol is in error.                  */
/*      The location in error is given by:  l_1 = s_2/s_1.              */
/*      The error contribution is given by: e_1 = s_1/l_1.              */

    if (alpha_mul(s_1,s_3) == alpha_mul(s_2,s_2))
    {

        if (alpha_mul(s_2,s_4) != alpha_mul(s_3,s_3)) return(3);
            l_1 = alpha_mul(s_2,alpha_inv(s_1));
            e_1 = alpha_mul(s_1,alpha_inv(l_1));

/*      Convert e_1 to polynomial format and subtract this from         */
/*      the received coefficient, thus giving the corrected value.      */

        sio_error[0][0] = l_1; sio_error[0][1] = log_to_lin[e_1];
        sio_error[1][0] = 0;   sio_error[1][1] = 0;

        if (l_1 >= max_block_len)
            return(3);

        *(received - l_1) ^= sio_error[0][1];

        return(1);
    }
```

```
/*      Otherwise two symbols are in error. Solve the equations:        */
/*                                                                      */
/*      s_1 * sig_2 + s_2 * sig_1 = s_3                                 */
/*      s_2 * sig_2 + s_3 * sig_1 = s_4                                 */
/*                                                                      */
/*      for sig_1 and sig_2 by Kramer's rule.                           */

    sig_1 = alpha_mul(
        alpha_add(alpha_mul(s_1,s_4),alpha_mul(s_2,s_3)),
        alpha_inv(alpha_add(alpha_mul(s_1,s_3),alpha_mul(s_2,s_2))));

    sig_2 = alpha_mul(
        alpha_add(alpha_mul(s_3,s_3),alpha_mul(s_2,s_4)),
        alpha_inv(alpha_add(alpha_mul(s_1,s_3),alpha_mul(s_2,s_2))));

/*      Now find the roots to the error locator polynomial equation:    */
/*                                                                      */
/*      x**2 + sig_1*x + sig_2 = 0                                      */
/*
/*      These roots, l_1, l_2 locate the symbols in error.              */

    sio_error[1][0] = l_2 = solve_poly(sig_1,sig_2);
    sio_error[0][0] = l_1 = alpha_add(l_2,sig_1);

/*      Solve the equations:                                            */
/*                                                                      */
/*      e_1 * a**l_1     + e_2 * a**l_2     = s_1                       */
/*      e_1 * a**(2*l_1) + e_2 * a**(2*l_2) = s_2                       */
/*                                                                      */
/*      for e_1, e_2, the error contributions, by Kramer's rule.        */
/*                                                                      */
/*      Use the relationship: a**l_1 + a**l_2 = sig_1 to simplify       */
/*      the result.                                                     */

    e_1 = alpha_mul(
        alpha_add(alpha_mul(s_1,l_2),s_2),
        alpha_inv(alpha_mul(l_1,sig_1)));

    e_2 = alpha_mul(
        alpha_add(alpha_mul(s_1,l_1),s_2),
        alpha_inv(alpha_mul(l_2,sig_1)));

    sio_error[0][1] = log_to_lin[e_1];
    sio_error[1][1] = log_to_lin[e_2];

    if (l_1 >= max_block_len || l_2 >= max_block_len)
        return(3);

    *(received - l_1) ^= sio_error[0][1];
    *(received - l_2) ^= sio_error[1][1];

/*      Treat adjacent errors as single errors                          */

    if (l_1 > l_2)
        return((l_1 - l_2 == 1) ? 1 : 2);
    else
        return((l_2 - l_1 == 1) ? 1 : 2);
```