Date: August 16, 02
To: T10 Technical Committee
From: Jim Coomes (jim.comes@seagate.com)
Subject: Bit order for scrambling and CRC

Below are C programs and equations to generate CRC and scrambled output for Annex A and C in the SAS draft. This revision includes an example scrambler diagram, changes sample scramble code to a serial shift register, scrambled data output for all zeros input, and example XOR equations for generating a 32 bit scramble value.

==

The following is an example C program that generates the value for the CRC field in frames. The inputs are the data dwords for the frame and the number of data dwords.

```
{   static unsigned long data_dwords[] = {0x06D0B992L,0x00B5DF59L,0x00000000L,
                                       0x00000000L,0x1234FFFFL,0x00000000L,
                                       0x00000000L,0x00000000L,0x00000000L,
                                       0x08000012L,0x01000000L,0x00000000L,
                                       0x00000000L};  /* example data dwords */

   unsigned long *frame; /* pointer to the data dwords  */
   unsigned long int length;   /* number of data dwords  */

   unsigned long calculate_crc( *frame, length);
   unsigned long crc;

   crc = calculate_crc(data_dwords,13);


unsigned long calculate_crc( *frame, length) /* returns crc value */
{    long poly = 0x04C11DB7L;
     unsigned long crc_gen, x;
     union {unsigned long lword; unsigned char byte[4];} b_access;
     static unsigned char xpose[] = {0x0,0x8,0x4,0xC,0x2,0xA,0x6,0xE,
                                    0x1,0x9,0x5,0xD,0x3,0xB,0x7,0xF};
     int i, j, fb;

     crc_gen = ~0;   /* seed generator with all ones */
     for (i=0; i<length; i++)
     {   x = *frame++; /* get word */
         b_access.lword = x; /* transpose bits in byte */
         for (j=0; j<4; j++)
         {   b_access.byte[j] = xpose[b_access.byte[j]>>4] |
xpose[b_access.byte[j]&0xF]<<4;
         }
         x = b_access.lword;

         for (j=0; j<32; j++) /* serial shift register implementation */
         {   fb = ((x & 0x80000000L)>0) ^ ((crc_gen & 0x80000000L)>0);
             x <<= 1;
             crc_gen <<= 1;
             if (fb)
                 crc_gen ^= poly;
```

```
        }
    }

    b_access.lword = crc_gen; /* transpose bits in CRC */
    for (j=0; j<4; j++)
    {   b_access.byte[j] = xpose[b_access.byte[j]>>4] |
xpose[b_access.byte[j]&0xF]<<4;
    }
    crc_gen = b_access.lword;

    return ~crc_gen; /* invert output */
}
}
```

===

These equations generate the 32 bit Cyclic Redundancy Check for frame transmission. The ^ symbol represents an XOR operation.

```
 crc00 = d00^d06^d09^d10^d12^d16^d24^d25^d26^d28^d29^d30^d31;
 crc01 = d00^d01^d06^d07^d09^d11^d12^d13^d16^d17^d24^d27^d28;
 crc02 = d00^d01^d02^d06^d07^d08^d09^d13^d14^d16^d17^d18^d24^d26^d30^d31;
 crc03 = d01^d02^d03^d07^d08^d09^d10^d14^d15^d17^d18^d19^d25^d27^d31;
 crc04 = d00^d02^d03^d04^d06^d08^d11^d12^d15^d18^d19^d20^d24^d25^d29^d30^d31;
 crc05 = d00^d01^d03^d04^d05^d06^d07^d10^d13^d19^d20^d21^d24^d28^d29;
 crc06 = d01^d02^d04^d05^d06^d07^d08^d11^d14^d20^d21^d22^d25^d29^d30;
 crc07 = d00^d02^d03^d05^d07^d08^d10^d15^d16^d21^d22^d23^d24^d25^d28^d29;
 crc08 = d00^d01^d03^d04^d08^d10^d11^d12^d17^d22^d23^d28^d31;
 crc09 = d01^d02^d04^d05^d09^d11^d12^d13^d18^d23^d24^d29;
 crc10 = d00^d02^d03^d05^d09^d13^d14^d16^d19^d26^d28^d29^d31;
 crc11 = d00^d01^d03^d04^d09^d12^d14^d15^d16^d17^d20^d24^d25^d26^d27^d28^d31;
 crc12 = d00^d01^d02^d04^d05^d06^d09^d12^d13^d15^d17^d18^d21^d24^d27^d30^d31;
 crc13 = d01^d02^d03^d05^d06^d07^d10^d13^d14^d16^d18^d19^d22^d25^d28^d31;
 crc14 = d02^d03^d04^d06^d07^d08^d11^d14^d15^d17^d19^d20^d23^d26^d29;
 crc15 = d03^d04^d05^d07^d08^d09^d12^d15^d16^d18^d20^d21^d24^d27^d30;
 crc16 = d00^d04^d05^d08^d12^d13^d17^d19^d21^d22^d24^d26^d29^d30;
 crc17 = d01^d05^d06^d09^d13^d14^d18^d20^d22^d23^d25^d27^d30^d31;
 crc18 = d02^d06^d07^d10^d14^d15^d19^d21^d23^d24^d26^d28^d31;
 crc19 = d03^d07^d08^d11^d15^d16^d20^d22^d24^d25^d27^d29;
 crc20 = d04^d08^d09^d12^d16^d17^d21^d23^d25^d26^d28^d30;
 crc21 = d05^d09^d10^d13^d17^d18^d22^d24^d26^d27^d29^d31;
 crc22 = d00^d09^d11^d12^d14^d16^d18^d19^d23^d24^d26^d27^d29^d31;
 crc23 = d00^d01^d06^d09^d13^d15^d16^d17^d19^d20^d26^d27^d29^d31;
 crc24 = d01^d02^d07^d10^d14^d16^d17^d18^d20^d21^d27^d28^d30;
 crc25 = d02^d03^d08^d11^d15^d17^d18^d19^d21^d22^d28^d29^d31;
 crc26 = d00^d03^d04^d06^d10^d18^d19^d20^d22^d23^d24^d25^d26^d28^d31;
 crc27 = d01^d04^d05^d07^d11^d19^d20^d21^d23^d24^d25^d26^d27^d29;
 crc28 = d02^d05^d06^d08^d12^d20^d21^d22^d24^d25^d26^d27^d28^d30;
 crc29 = d03^d06^d07^d09^d13^d21^d22^d23^d25^d26^d27^d28^d29^d31;
 crc30 = d04^d07^d08^d10^d14^d22^d23^d24^d26^d27^d28^d29^d30;
 crc31 = d05^d08^d09^d11^d15^d23^d24^d25^d27^d28^d29^d30^d31;
```

===

The following is an example of the CRC calculation for a 6 byte Read command IU.

INFORMATION UNIT TYPE 06h

HASHED DESTINATION DEVICE NAME for 500107534F0CFC88h	D0B992h
HASHED SOURCE DEVICE NAME for 50010B92B3CBF639h	B5DF59h
TIMEOUT	0
NUMBER OF FILL BYTES	0
COMMAND ID	0
TAG	1234h
TARGET PORT TRANSFER TAG	FFFFh
INFORMATION UNIT (CMD)	
LUN	0
TASK ATTRIBRUTE	0
ADDITIONAL CDB LENGTH	0
CDB (6 byte read block 12h)	080000120100h

Data dwords

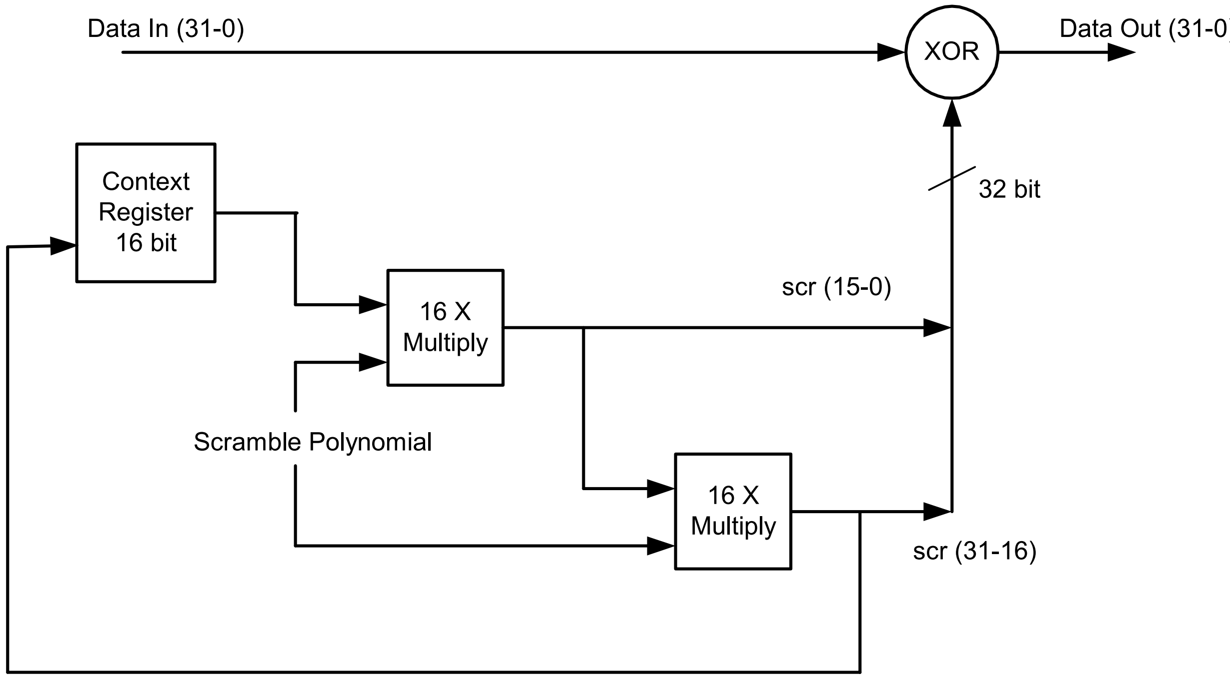
```
06D0B992h
00B5DF59h
00000000h
00000000h
1234FFFFh
00000000h
00000000h
00000000h
00000000h
08000012h
01000000h
00000000h
00000000h
```

CRC = 3F4F1C26h

==

Data scrambler implementation example

Figure xx shows an example of a data scrambler. This example generates the value to XOR with the dword input with two 16 bit parallel multipliers. 16 bits wide is the maximum width for the multiplier as the generating polynomial is 16 bits.




```
===========================================================================
The following is an example C program that generates the scrambled data dwords
for transmission. The inputs are the data dword to scramble and control
indication to reinitialize the residual value following an SOF.

The generator polynomial specified is:
        16   15   13   4
G(x) = x  + x  + x  + x + 1

For parallelized versions of the scrambler, the initialized value is selected to
produce a first dword output of 0xC2D2768D for a dword input of all zeros.


#include <stdio.h>
//--------------------------------------------------------------------------------
unsigned long scramble(bool reset, unsigned long dword);
void main1(void)
{ FILE *f = fopen("crc_report.txt","w");
  if (f)
  { for (int i=0; i<12; i++)
      fprintf(f," %08X \n",scramble(i==0, 0));
    fclose(f);
  }
}
/* sample output (if input == 0)
0xC2D2768D
0x1F26B368
0xA508436C
0x3452D354
0x8A559502
0xBB1ABE1B
0xFA56B73D
```

```
0x53F60B1B
0xF0809C41
0x747FC34A
0xBE865291
0x7A6FA7B6
*/

#define poly 0xA011
unsigned long scramble(bool reset, unsigned long dword)
{ static unsigned short scramble;
  if (reset)
     scramble = 0xFFFF;
  for (int i=0; i<32; i++)  /* serial shift register implementation */
  { dword ^= scramble&0x8000?1<<i:0;
    scramble = (scramble<<1) ^ (scramble&0x8000? poly : 0);
  }
  return dword;
}==========================================================================
```

These equations generate the scrambled bytes to XOR with dwords before transmission and dword reception to recover the original data. The ^ symbol represents an XOR operation. The initialized value for d[15-0] is 0xF0F6 in this example.

```
scr[31]=d[12]^d[10]^d[7]^d[3]^d[1]^d[0];
scr[30]=d[15]^d[14]^d[12]^d[11]^d[9]^d[6]^d[3]^d[2]^d[0];
scr[29]=d[15]^d[13]^d[12]^d[11]^d[10]^d[8]^d[5]^d[3]^d[2]^d[1];
scr[28]=d[14]^d[12]^d[11]^d[10]^d[9]^d[7]^d[4]^d[2]^d[1]^d[0];
scr[27]=d[15]^d[14]^d[13]^d[12]^d[11]^d[10]^d[9]^d[8]^d[6]^d[1]^d[0];
scr[26]=d[15]^d[13]^d[11]^d[10]^d[9]^d[8]^d[7]^d[5]^d[3]^d[0];
scr[25]=d[15]^d[10]^d[9]^d[8]^d[7]^d[6]^d[4]^d[3]^d[2];
scr[24]=d[14]^d[9]^d[8]^d[7]^d[6]^d[5]^d[3]^d[2]^d[1];
scr[23]=d[13]^d[8]^d[7]^d[6]^d[5]^d[4]^d[2]^d[1]^d[0];
scr[22]=d[15]^d[14]^d[7]^d[6]^d[5]^d[4]^d[1]^d[0];
scr[21]=d[15]^d[13]^d[12]^d[6]^d[5]^d[4]^d[0];
scr[20]=d[15]^d[11]^d[5]^d[4];
scr[19]=d[14]^d[10]^d[4]^d[3];
scr[18]=d[13]^d[9]^d[3]^d[2];
scr[17]=d[12]^d[8]^d[2]^d[1];
scr[16]=d[11]^d[7]^d[1]^d[0];
scr[15]=d[15]^d[14]^d[12]^d[10]^d[6]^d[3]^d[0];
scr[14]=d[15]^d[13]^d[12]^d[11]^d[9]^d[5]^d[3]^d[2];
scr[13]=d[14]^d[12]^d[11]^d[10]^d[8]^d[4]^d[2]^d[1];
scr[12]=d[13]^d[11]^d[10]^d[9]^d[7]^d[3]^d[1]^d[0];
scr[11]=d[15]^d[14]^d[10]^d[9]^d[8]^d[6]^d[3]^d[2]^d[0];
scr[10]=d[15]^d[13]^d[12]^d[9]^d[8]^d[7]^d[5]^d[3]^d[2]^d[1];
scr[9]=d[14]^d[12]^d[11]^d[8]^d[7]^d[6]^d[4]^d[2]^d[1]^d[0];
scr[8]=d[15]^d[14]^d[13]^d[12]^d[11]^d[10]^d[7]^d[6]^d[5]^d[1]^d[0];
scr[7]=d[15]^d[13]^d[11]^d[10]^d[9]^d[6]^d[5]^d[4]^d[3]^d[0];
scr[6]=d[15]^d[10]^d[9]^d[8]^d[5]^d[4]^d[2];
scr[5]=d[14]^d[9]^d[8]^d[7]^d[4]^d[3]^d[1];
scr[4]=d[13]^d[8]^d[7]^d[6]^d[3]^d[2]^d[0];
scr[3]=d[15]^d[14]^d[7]^d[6]^d[5]^d[3]^d[2]^d[1];
scr[2]=d[14]^d[13]^d[6]^d[5]^d[4]^d[2]^d[1]^d[0];
scr[1]=d[15]^d[14]^d[13]^d[5]^d[4]^d[1]^d[0];
scr[0]=d[15]^d[13]^d[4]^d[0];
```

```
================================================================================

The following examples of the scrambled data output.

Data                scrambled
Dword               output

06D0B992h --> C402CF1Fh
00B5DF59h --> 1F936C31h
00000000h --> A508436Ch
00000000h --> 3452D354h
1234FFFFh --> 98616AFDh
00000000h --> BB1ABE1Bh
00000000h --> FA56B73Dh
00000000h --> 53F60B1Bh
00000000h --> F0809C41h
08000012h --> 7C7FC358h
01000000h --> BF865291h
00000000h --> 7A6FA7B6h
00000000h --> 3163E6D6h
3F4F1C26h --> CF79E22Ah

Data                scrambled
Dword               output

00000000h  --> C2D2768Dh
00000000h  --> 1F26B368h
00000000h  --> A508436Ch
00000000h  --> 3452D354h
00000000h  --> 8A559502h
00000000h  --> BB1ABE1Bh
00000000h  --> FA56B73Dh
00000000h  --> 53F60B1Bh
00000000h  --> F0809C41h
00000000h  --> 747FC34Ah
00000000h  --> BE865291h
00000000h  --> 7A6FA7B6h
00000000h  --> 3163E6D6h
B00F2BCCh  --> 4039D5C0h    (CRC dword)
```