TYPICAL BiSS CRC POLYNOMIALS

CRC Polynomials Hex.	CRC Polynomials Polynomial	Max. Data Length	Hamming Distance	Application
0x13	$x^4 + x^1 + x^0$	up to 11 Bit	3	Control communication (register communication)
0x25	$x^5 + x^2 + x^0$	up to 26 Bit	3	Process data (e.g. temperature)
0x43	$x^6 + x^1 + x^0$	up to 57 Bit	3	Process data (e.g. position)
0x190D9	$x^{16} + x^{15} + x^{12} + x^7 + x^6 + x^4 + x^3 + x^0$	up to 64 Bit	6	Process data (e.g. position in safety systems)

Table 1: Typical *BiSS* CRC Polynomials

CRC IN PROCESS DATA COMMUNICATION

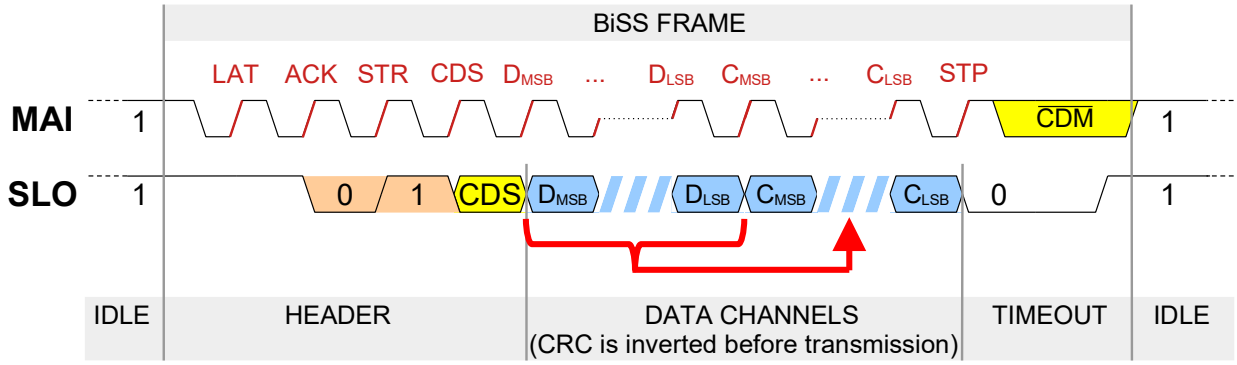

Figure 1: Process data and CRC in *BiSS* frame (point-to-point connection)

Typically, the process data communication is secured by a 6-bit cyclic redundancy check C_{MSB} ... C_{LSB} (polynomial 0x43, start value 0x00). All data bits of the data channel D_{MSB} ... D_{LSB} (incl. sensor data, status bits, sign-of-life counter) are considered for CRC calculation. The 6-bit CRC is inverted before transmission.

In safety-critical applications with *BiSS* Safety, two redundant position words are transmitted

- Control Position Word (CPW)
- Safety Position Word (SPW)

The CPW is protected by a 6-bit CRC (polynomial 0x43), the SPW is protected by a 16-bit CRC (polynomial 0x190D9). Both CRCs are inverted before transmission.

Copyright © 2012, 2023 iC-Haus

CRC IN REGISTER COMMUNICATION

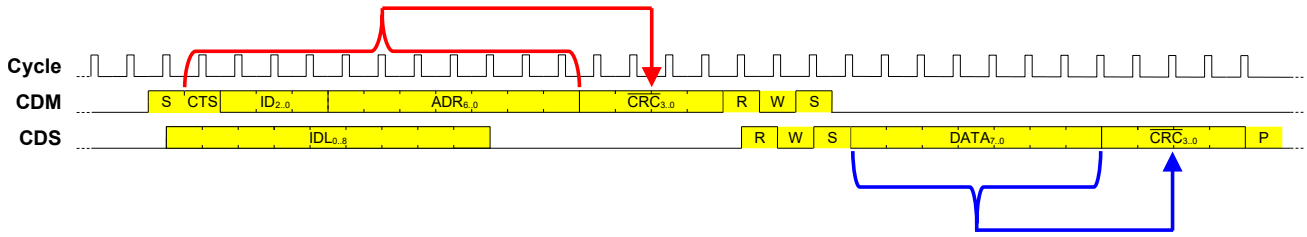

Figure 2: Read register access

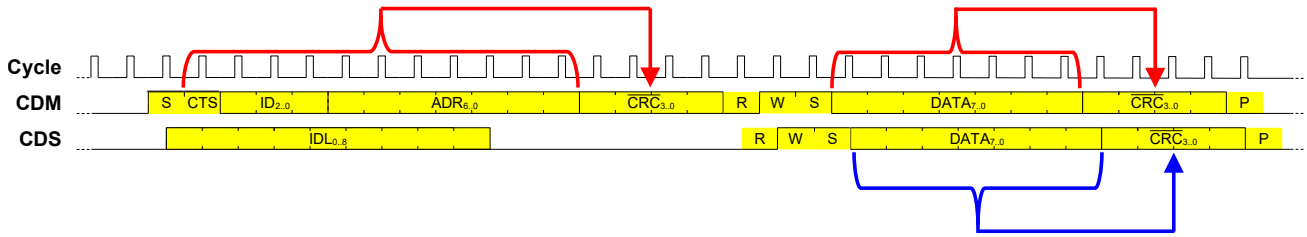

Figure 3: Write register access

The register communication is secured by a 4-bit CRC (polynomial 0x13, start value 0x00). In a single register access (read or write) two CRCs are transmitted and evaluated. The 4-bit CRCs are inverted before transmission.

For the first CRC transmitted by the master, the CRC calculation considers:

- CTS
- $ID_{2:0}$
- $ADR_{6:0}$

For the second CRC transmitted by the master during register write operation, the CRC calculation considers:

- $DATA_{7:0}$

For the response by the slave, the CRC calculation considers:

- $DATA_{7:0}$

CRC CODE EXAMPLE: n BIT CRC

```
//#############################################################################################

#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*****************************************************************************
* Function : generates the CRC sum from a given data set (pstrDataSetIn)
*            and CRC polynomial (pstrCRCPolyIn).
*****************************************************************************/
char *pstrCRCGEN (char *pstrDataSetIn, char *pstrCRCPolyIn)
{
    int i,j;
    char            *pstrDataSet = strdup(pstrDataSetIn),
                    *pstrCRCPoly = strdup(pstrCRCPolyIn);
    unsigned short  usDatLen = strlen(pstrDataSet),
                    usCRCPolyLen = strlen(pstrCRCPoly),
                    usCRCLen = usCRCPolyLen-1;
    char            *pstrCRC = malloc(usCRCLen);
    char            cExOr = '0';

    pstrDataSet  = strrev(pstrDataSet);
    pstrCRCPoly = strrev(pstrCRCPoly);


    // Initialize pstrCRC with '0'
    for (i = usCRCLen-1; i >= 0; i--)
        pstrCRC[i] = '0';

    // Calculate pstrCRC
    for (i = usDatLen-1; i >= 0; i--)
    {
        switch (pstrDataSet[i])
        {
            case '0':
                if (pstrCRC[usCRCLen-1] == '0')
                    cExOr = '0';
                else
                    cExOr = '1';
                break;

            case '1':
                if (pstrCRC[usCRCLen-1] == '1')
                    cExOr = '0';
                else
                    cExOr = '1';
                break;

            default:
                exit (1);
        }

        //
        for (j = usCRCLen-1; j > 0; j--)
        {
            if (pstrCRCPoly[j] == '1')
            {
                if (pstrCRC[j-1] == cExOr)
                    pstrCRC[j] = '0';
                else
                    pstrCRC[j] = '1';
            }
            else
                pstrCRC[j] = pstrCRC[j-1];
        }
        pstrCRC[0] = cExOr;  // Bit 0
    }
    return (pstrCRC);
}
```

```
77  // Main function (prints calculated CRC)
78  char *main(int argc, char *argv[])
79  {
80      int  i = 0;
81      char *pstrCRCWert = NULL;
82
83      if (argc < 2)
84      {
85          printf("Call:␣CRCGen␣<data>␣<CRC-Poly>\n");
86          printf("␣␣␣␣␣␣␣␣<data>␣binary␣data,␣the␣crc-code␣should␣be␣created␣for\n");
87          printf("␣␣␣␣␣␣␣␣<CRC-Poly>␣binary␣Polynomial,␣used␣to␣create␣the␣CRC\n");
88          return (0);
89      }
90      else
91      {
92          pstrCRCWert = pstrCRCGEN (argv[1], argv[2]);
93          printf ("\n\n");
94          printf ("data␣␣␣%s\n", argv[1]);
95          printf ("CRC␣Polynomial␣␣%s\n", argv[2]);
96          printf ("\nergibt␣CRC␣␣");
97          for (i = strlen(argv[2])-2; i >= 0; i--)
98            printf ("%c", pstrCRCWert[i]);
99          printf ("\n\n\n");
100     }
101     return (pstrCRCWert);
102 }
103
104
105 //#############################################################################################
```

The calculated CRC needs to be inverted before transmission