# BiSS Interface
AN21: CRC VERIFICATION FOR MICROCONTROLLER

## CONTENTS

## CRC VERIFICATION

A CRC can be used to protect a set of bits to be sent. And a CRC can be used to verify the integrity of a set of bits that were received. Popular applications are single cycle data (SCD) CRC verification and device configuration protection/verification.

There are different algorithmic approaches to calculate a CRC:

- bit-by-bit CRC calculation
- table-driven CRC calculation
- each with and without augmented zero bytes

The user decides which is best for his application due to performance and timing requirements.

**ATTENTION!**
The listed software (code and its documentation) is provided by iC-Haus GmbH "AS IS".
See Software disclaimer on page 9.

## CONFIGURATION FOR 6 BIT CRC

Listing 1: Parameter for 6 bit CRC

```
//
// CRC Parameter for 6 bit CRC used by BiSS (for example iC-LGC)
// changed by iC-Haus 2013
//
const int order = 6;
const unsigned long polynom = 0x03; // => 0x43 without leading 1
const int direct = 1;
const unsigned long crcinit = 0x0000;
const unsigned long crcxor = 0x003f;
const int refin = 0;
const int refout = 0;
```

If a CRC start value is used the value **crcinit** must be set to the correct value.

## CONFIGURATION FOR 16 BIT CRC

For the 16 bit BiSS C CRC used by iC-LGC the CRC parameters must be changed (default values are for CRC-32).

Listing 2: Parameter for 16 bit CRC

```
//
// CRC Parameter for 16 bit CRC used by BiSS (for example iC-LGC)
// changed by iC-Haus 2013
//

const int order = 16;
const unsigned long polynom = 0x90d9;
const int direct = 1;
const unsigned long crcinit = 0x0000;
const unsigned long crcxor = 0xffff;
const int refin = 0;
const int refout = 0;
```

If a CRC start value is used the value **crcinit** must be set to the correct value.

## EXAMPLE 6 BIT CRC

This example shows the output of the C algorithms for a BiSS C SCD data frame with 12 bits (Data "010011010101") without the calculated 6 bit CRC value. For this example the data section of the source code was changed like this:

Listing 3: Data string for 6 bit CRC example

```
//
// Data string for 12 bit BiSS C data frame
// changed by iC-Haus 2013
//
const unsigned char string[] = {0x04, 0xD5};
```

```
CRC test
--------

Parameters:
 polynom              :  0x3
 order                :  6
 crcinit              :  0x0 direct, 0x0 nondirect
 crcxor               :  0x3F
 refin                :  0
 refout               :  0

 data string          :  '0x04' '0xD5' (2 bytes)

Results:
 crc bit by bit       :  0x1c
 crc bit by bit fast  :  0x1c
```

## EXAMPLE 16 BIT CRC

This example shows the output of the C algorithms for a BiSS C SCD data frame with 40 bits (12 bit MT + 20 bit ST + 1 bit Err + 1 bit Warn + 6 bit LC) without the calculated 16 bit CRC value. For this example the data section of the source code was changed like this:

Listing 4: Data string for 16 bit CRC example

```c
// Data character string

//const unsigned char string[] = {"123456789"};
const unsigned char string[] = {0xD9, 0xCF, 0xE0, 0xC0, 0xDA};
```

```
CRC test
---------

Parameters:
 polynom              :  0x90d9
 order                :  16
 crcinit              :  0x0 direct, 0x0 nondirect
 crcxor               :  0xFFFF
 refin                :  0
 refout               :  0

 data string          :  '0xD9' , '0xCF' , '0xE0' , '0xC0' , '0xDA' (5 bytes)

Results:
 crc bit by bit       :  0x5F29
 crc bit by bit fast  :  0x5F29
 crc table            :  0x5F29
 crc table fast       :  0x5F29
```

## CONFIGURATION CRC

A CRC can be used to verify the integrity of a bigger set of bits. Some devices require a CRC to protect/verify a configuration subset.

**Example application: iC-LGC CRC of configuration data:**

| **REGISTER ASSIGNMENT** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Addr | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| **CRC of configuration data** | | | | | | | | |
| 0x32 | CRCCFG(15:8): CRC of configuration data, high byte | | | | | | | |
| 0x33 | CRCCFG(7:0): CRC of configuration data, low byte | | | | | | | |

Table 1: Register layout

**ATTENTION!**
A checksum is not a CRC and just adds all address or byte values.
Some configuration data and BANK wide content are only checksum protected.

## APPENDIX

Listing 5: Example source code for 16 bit CRC

```
// -----------------------------------------------------------------------------
// CRC test
// changed by iC-Haus 2013
// -----------------------------------------------------------------------------

// includes:

#include <string.h>
#include <stdio.h>


// CRC parameters (default values are for CRC-32):

//const int order = 32;
//const unsigned long polynom = 0x4c11db7;
//const int direct = 1;
//const unsigned long crcinit = 0xffffffff;
//const unsigned long crcxor = 0xffffffff;
//const int refin = 1;
//const int refout = 1;

//
// CRC Parameter for 16 bit CRC used by BiSS (for example iC-LGC)
// changed by iC-Haus 2013
//

const int order = 16;
const unsigned long polynom = 0x90d9;
const int direct = 1;
const unsigned long crcinit = 0x0000;
const unsigned long crcxor = 0xffff;
const int refin = 0;
const int refout = 0;


// ’order’ [1..32] is the CRC polynom order, counted without the leading ’1’ bit
// ’polynom’ is the CRC polynom without leading ’1’ bit
// ’direct’ [0,1] specifies the kind of algorithm: 1=direct, no augmented zero bits
// ’crcinit’ is the initial CRC value belonging to that algorithm
// ’crcxor’ is the final XOR value
// ’refin’ [0,1] specifies if a data byte is reflected before processing (UART) or not
// ’refout’ [0,1] specifies if the CRC will be reflected before XOR


//
// Data string for 40 bit BiSS C data frame
// changed bei iC-Haus 2013
//

// Data character string

//const unsigned char string[] = {"123456789"};
const unsigned char string[] = {0xD9, 0xCF, 0xE0, 0xC0, 0xDA};


// internal global values:

unsigned long crcmask;
unsigned long crchighbit;
unsigned long crcinit_direct;
unsigned long crcinit_nondirect;
unsigned long crctab[256];


// subroutines
```

```
unsigned long reflect (unsigned long crc, int bitnum) {

 // reflects the lower 'bitnum' bits of 'crc'

 unsigned long i, j=1, crcout=0;

 for (i=(unsigned long)1<<(bitnum-1); i; i>>=1) {
  if (crc & i) crcout|=j;
  j<<= 1;
 }
 return (crcout);
}



void generate_crc_table() {

 // make CRC lookup table used by table algorithms

 int i, j;
 unsigned long bit, crc;

 for (i=0; i<256; i++) {

  crc=(unsigned long)i;
  if (refin) crc=reflect(crc, 8);
  crc<<= order-8;

  for (j=0; j<8; j++) {

   bit = crc & crchighbit;
   crc<<= 1;
   if (bit) crc^= polynom;
  }

  if (refin) crc = reflect(crc, order);
  crc&= crcmask;
  crctab[i]= crc;
 }
}



unsigned long crctablefast (unsigned char* p, unsigned long len) {

 // fast lookup table algorithm without augmented zero bytes, e.g. used in pkzip.
 // only usable with polynom orders of 8, 16, 24 or 32.

 unsigned long crc = crcinit_direct;

 if (refin) crc = reflect(crc, order);

 if (!refin) while (len--) crc = (crc << 8) ^ crctab[ ((crc >> (order-8)) & 0xff) ^ *p++];
 else while (len--) crc = (crc >> 8) ^ crctab[ (crc & 0xff) ^ *p++];

 if (refout^refin) crc = reflect(crc, order);
 crc^= crcxor;
 crc&= crcmask;

 return(crc);
}



unsigned long crctable (unsigned char* p, unsigned long len) {

 // normal lookup table algorithm with augmented zero bytes.
 // only usable with polynom orders of 8, 16, 24 or 32.

 unsigned long crc = crcinit_nondirect;
 if (refin) crc = reflect(crc, order);

 if (!refin) while (len--) crc = ((crc << 8) | *p++) ^ crctab[ (crc >> (order-8))  & 0xff];
 else while (len--) crc = ((crc >> 8) | (*p++ << (order-8))) ^ crctab[ crc & 0xff];

 if (!refin) while (++len < order/8) crc = (crc << 8) ^ crctab[ (crc >> (order-8))  & 0xff];
 else while (++len < order/8) crc = (crc >> 8) ^ crctab[crc & 0xff];

 if (refout^refin) crc = reflect(crc, order);
 crc^= crcxor;
 crc&= crcmask;

 return(crc);
}



unsigned long crcbitbybit(unsigned char* p, unsigned long len) {

 // bit by bit algorithm with augmented zero bytes.
 // does not use lookup table, suited for polynom orders between 1...32.

 unsigned long i, j, c, bit;
 unsigned long crc = crcinit_nondirect;

 for (i=0; i<len; i++) {

  c = (unsigned long)*p++;
  if (refin) c = reflect(c, 8);

  for (j=0x80; j; j>>=1) {

   bit = crc & crchighbit;
   crc<<= 1;
   if (c & j) crc|= 1;
   if (bit) crc^= polynom;
  }
 }

 for (i=0; i<order; i++) {

  bit = crc & crchighbit;
  crc<<= 1;
  if (bit) crc^= polynom;
 }

 if (refout) crc=reflect(crc, order);
 crc^= crcxor;
 crc&= crcmask;

 return(crc);
}



unsigned long crcbitbybitfast(unsigned char* p, unsigned long len) {

 // fast bit by bit algorithm without augmented zero bytes.
 // does not use lookup table, suited for polynom orders between 1...32.

 unsigned long i, j, c, bit;
 unsigned long crc = crcinit_direct;

 for (i=0; i<len; i++) {

  c = (unsigned long)*p++;
  if (refin) c = reflect(c, 8);

  for (j=0x80; j; j>>=1) {

   bit = crc & crchighbit;
   crc<<= 1;
   if (c & j) bit^= crchighbit;
   if (bit) crc^= polynom;
  }
 }

 if (refout) crc=reflect(crc, order);
 crc^= crcxor;
 crc&= crcmask;

 return(crc);
}



int main() {

 // test program for checking four different CRC computing types that are:
 // crcbit(), crcbitfast(), crctable() and crctablefast(), see above.
 // parameters are at the top of this program.
 // Result will be printed on the console.

 int i;
 unsigned long bit, crc;


 // at first, compute constant bit masks for whole CRC and CRC high bit

 crcmask = ((((unsigned long)1<<(order-1))-1)<<1)|1;
 crchighbit = (unsigned long)1<<(order-1);


 // check parameters

 if (order < 1 || order > 32) {
  printf("ERROR,␣invalid␣order,␣it␣must␣be␣between␣1..32.\n");
  return(0);
 }

 if (polynom != (polynom & crcmask)) {
  printf("ERROR,␣invalid␣polynom.\n");
  return(0);
 }

 if (crcinit != (crcinit & crcmask)) {
  printf("ERROR,␣invalid␣crcinit.\n");
  return(0);
 }

 if (crcxor != (crcxor & crcmask)) {
  printf("ERROR,␣invalid␣crcxor.\n");
  return(0);
 }


 // generate lookup table

 generate_crc_table();

  printf("␣CRCTAB␣=␣");
 for (i=0;i<256;i++)
    printf("%X, ", crctab[i]);
  printf("\n");

 // compute missing initial CRC value

 if (!direct) {

  crcinit_nondirect = crcinit;
  crc = crcinit;
  for (i=0; i<order; i++) {
    bit = crc & crchighbit;
    crc<<= 1;
    if (bit) crc^= polynom;
   }
  crc&= crcmask;
  crcinit_direct = crc;
 }

else {

  crcinit_direct = crcinit;
  crc = crcinit;
  for (i=0; i<order; i++) {

    bit = crc & 1;
    if (bit) crc^= polynom;
    crc >>= 1;
    if (bit) crc|= crchighbit;
   }
  crcinit_nondirect = crc;
 }


// call CRC algorithms using the CRC parameters above and print result to the console

printf("\n");
printf("CRC␣test\n");
printf("--------\n");
printf("\n");
printf("Parameters:\n");
printf("\n");
printf("␣polynom␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣0x%x\n", polynom);
printf("␣order␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣%d\n", order);
printf("␣crcinit␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣0x%x␣direct,␣0x%x␣nondirect\n", crcinit_direct, crcinit_nondirect);
printf("␣crcxor␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣0x%x\n", crcxor);
printf("␣refin␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣%d\n", refin);
printf("␣refout␣␣␣␣␣␣␣␣␣␣␣␣␣␣:␣␣%d\n", refout);
printf("\n");
printf("␣data␣string␣␣␣␣␣␣␣␣␣:␣␣'%s'␣(%d␣bytes)\n", string, strlen(string));
printf("\n");
printf("Results:\n");
printf("\n");

printf("␣crc␣bit␣by␣bit␣␣␣␣␣␣:␣␣0x%x\n", crcbitbybit((unsigned char *)string, strlen(string)));
printf("␣crc␣bit␣by␣bit␣fast␣:␣␣0x%x\n", crcbitbybitfast((unsigned char *)string, strlen(string)));
if (!(order&7)) printf("␣crc␣table␣␣␣␣␣␣␣␣␣␣␣:␣␣0x%x\n", crctable((unsigned char *)string, strlen(
     string)));
if (!(order&7)) printf("␣crc␣table␣fast␣␣␣␣␣␣:␣␣0x%x\n", crctablefast((unsigned char *)string, strlen
     (string)));

return(0);
}
```

## RELATED DOCUMENTS

- BiSS website -
  → http://www.ichaus.de/product/BiSS Interface
- BiSS C protocol description -
  → http://www.ichaus.de/BiSS-C_en
- iC-LGC Product page -
  → http://www.ichaus.de/iC-LGC
- MB3U Product page -
  → http://www.ichaus.de/MB3U
- MB4U Product page - Specification -
  → http://www.ichaus.de/MB4U
- MB5U Product page - Specification -
  → http://www.ichaus.de/MB5U
- BiSS Reader GUI - GUI software for Windows PC -
  → http://www.ichaus.de/BiSS_gui_rte
- BiSS Interface DLL - Library Description -
  → http://www.ichaus.de/biss1sl_interface

## REVISION HISTORY

| Rel. | Rel. Date* | Chapter | Modification | Page |
|---|---|---|---|---|
| A1 | 2017-04-12 | | Initial release | all |

iC-Haus expressly reserves the right to change its products and/or specifications. An Infoletter gives details as to any amendments and additions made to the relevant current specifications on our internet website www.ichaus.com/infoletter and is automatically generated and shall be sent to registered users by email. Copying – even as an excerpt – is only permitted with iC-Haus' approval in writing and precise reference to source.

The data specified is intended solely for the purpose of product description and shall represent the usual quality of the product. In case the specifications contain obvious mistakes e.g. in writing or calculation, iC-Haus reserves the right to correct the specification and no liability arises insofar that the specification was from a third party view obviously not reliable. There shall be no claims based on defects as to quality in cases of insignificant deviations from the specifications or in case of only minor impairment of usability.
No representations or warranties, either expressed or implied, of merchantability, fitness for a particular purpose or of any other nature are made hereunder with respect to information/specification or the products to which information refers and no guarantee with respect to compliance to the intended use is given. In particular, this also applies to the stated possible applications or areas of applications of the product.

iC-Haus products are not designed for and must not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death *(Safety-Critical Applications)* without iC-Haus' specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems. iC-Haus products are not designed nor intended for use in military or aerospace applications or environments or in automotive applications unless specifically designated for such use by iC-Haus.
iC-Haus conveys no patent, copyright, mask work right or other trade mark right to this product. iC-Haus assumes no liability for any patent and/or other trade mark rights of a third party resulting from processing or handling of the product and/or any other use of the product.

Software and its documentation is provided by iC-Haus GmbH or contributors "AS IS" and is subject to the ZVEI General Conditions for the Supply of Products and Services with iC-Haus amendments and the ZVEI Software clause with iC-Haus amendments (www.ichaus.com/EULA).

* Release Date format: *YYYY-MM-DD*