The Cyclic Redundancy Check (CRC) for AX.25

Bill Newhall, KB2BRD
billnewhall@yahoo.com
Boulder, Colorado

1 Scope

This document describes basic implementation and testing of the cyclic redundancy check (CRC) for the AX.25 amateur packet radio protocol (also defined by CRC-CCITT and in ISO 3309).

2 Overview of the AX.25 CRC

The AX.25 CRC is used by amateur radio terminal node controllers (TNC) to produce a frame-check sequence (FCS) for error detection. A basic implementation of the CRC for AX.25 uses a shift register and XOR operations. The implementation provided here is practical and illustrative but is not optimized with respect to processing speed.

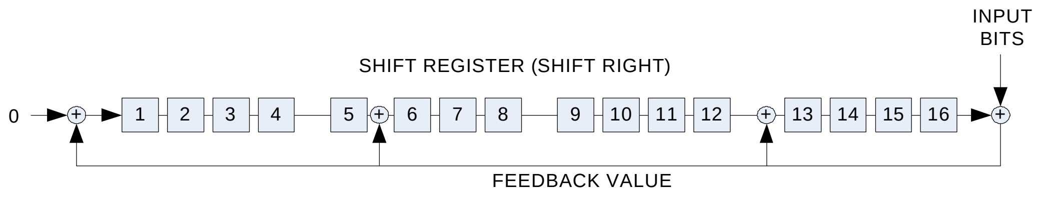
The AX.25 CRC is described by the characteristics given in Table 1.

Table 1. Characteristics of the AX.25 CRC.

Names associate with the AX.25 CRC protocol	CRC-CCITT, CRC-16-X25, CRC-16-CCITT
Degree of polynomial	16
Generator polynomial	G(x) = $x^{16}+x^{12}+x^5+1$
Polynomial in hex notation	0x1021
Bit order	Reflected input byte (LSB of byte sent first)
Initial value	0xFF
Output XOR mask	0xFF (bitwise-invert shift register to produce frame check sequence)

3 Implementation of the AX.25 CRC with a Shift Register

Figure 1 shows a shift-register implementation of the CRC algorithm for AX.25. This CRC generator uses the CRC-CCITT generator polynomial 0x1021. The 16-bit shift register is initialized with all ones (0xFFFF). The each numbered block is a stage of the shift register and holds one bit. The "+" circles indicate the XOR operation. The shift register bit values shift to the right. XOR operations are applied between shift register stages as indicated in the figure; for example, during a shift, the value of stage 5 is XORed with the feedback value, and the XOR result is stored in stage 6. The value shifted into stage 1 is a zero XORed with the feedback value; this is equivalent to shifting the feedback value into stage 1, but XORing with a zero is illustrative of the method used in software where a bitwise-XOR mask is applied to the shift register after the shift is performed.



NOTES:
INITIALIZE SHIFT REGISTER WITH ALL ONES
FCS IS BITWISE INVERSION OF SHIFT REGISTER AFTER LAST INPUT BIT

Figure 1. Implementation of the CRC-CCITT for AX.25.

4 Implementation of the AX.25 CRC in MATLAB

The following is a function that implements the AX.25 CRC algorithm in MATLAB. This function implements the AX.25 algorithm as shown in Figure 1.

```
% CRC_CCITT_Generator.m
%
% Generates the CRC for a sequence of bits based on the CCITT-CRC
% (CRC-16-CCITT) cyclic redundancy check.  The AX.25 amateur packet
% radio protocol uses the CCITT-CRC.
%
% function FCS = CRC_CCITT_Generator( Bits )
%
% Bits is input message bits [ First bit, ..., Last bit ]
% FCS is frame check sequence [ MSB, ..., LSB ].
%
% Rev. 11/23/2008 (W. Newhall, billnewhall@yahoo.com)

function FCS = CRC_CCITT_Generator( Bits )

G = [ 0 0 0 1  0 0 0 0  0 0 1 0  0 0 0 1 ];     % Generator polynomial for CCITT-CRC
% 0x        1        0        2        1        % CCITT_CRColynomial is 0x1021 in hex

r = 16;                         % Length of shift register
GFlip = fliplr( G );            % Flip the polynomial for the XOR operation in the loop
N_bits = length( Bits );        % Number of bits in message
SR = ones( 1, r );              % Initialize shift register (SR) to all ones

% Loop through all bits; shift bits into shift register
for( n_bit = 1 : N_bits )
    OutBit = SR( r );                   % Bit shifted out of shift register
    SR = [ 0, SR( 1 : ( r-1 ) ) ];  % Shift the register to the right and shift a zero in

    % Method 1
    XORMask = xor( Bits( n_bit ), OutBit ) * GFlip; % Input bit XORed w/ shifted-out bit
    SR = xor( SR, XORMask );        % Apply the XOR mask to the shift register

    % Alternate (Method 2)
    % if( OutBit ~= Bits( n_bit ) )         % Apply XOR mask only if shifted-
    %    SR = xor( SR, GFlip( 1 : r ) );    % out bit does not equal input bit.
    % end

    % Uncomment to display contents of SR for each bit:
    %  disp( num2str( ( SR ) ) );
end

% Bitwise-invert and flip the SR to get FCS with MSB as first element
FCS = fliplr( -SR+1 );

% disp( num2str( ( FCS ) ) ); % Display the SR
```

5 Testing the AX.25 CRC MATLAB Algorithm

The MATLAB CRC algorithm was tested against known input and output values for the CRC calculation. Three test cases are given here:

- Input and output values using an actual AX.25 frame
- Input, output, and intermediate values using a 24-bit sequence
- Input and output values using a common test sequence

5.1 Test Values Using an AX.25 Frame

An unnumbered information frame was sent using a PacComm PicoPacket TNC. The output waveform (audio FSK) of the TNC was captured, demodulated, and decoded on a PC using MATLAB. The decoded fields are as follows:

```
       FlagStart: 126
         Address: {'CQ-0'  'KB2BRD-2'}
            SSID: [0 2]
              CH: [0 1]
    ControlField: [1 1 0 0 0 0 0 0]
    InfoIncluded: 1
       FrameType: 'Unnumbered Frame'
      SupervFunc: 'Unnumbered Information-UI'
       PollFinal: 0
        Protocol: 'No Layer 3 Implemented'
            Info: 'A'
             FCS: [0 1 0 1 0 1 1 0 0 1 0 1 1 1 1 1]
       FrameBits: [1x176 double]
           Error: 0
       ErrorDesc: 'No error'
```

This frame was produced using the TNC by putting the TNC into “converse” mode, pressing the character “A”, and pressing *enter*. The FCS is calculated by applying the CRC algorithm to the bits between the AX.25 start flag (01111110) and the FCS field that is appended to the end of the frame before the end flag (see [1] for AX.25 frame construction). The demodulated bits from the TNC used for the FCS calculation were:

```
BitsForFCSCheck =
  Columns 1 through 13
     0     1     1     0     0     0     0     1     0     1     0     0     0
  Columns 14 through 26
     1     0     1     0     0     0     0     0     0     1     0     0     0
  Columns 27 through 39
     0     0     0     0     1     0     0     0     0     0     0     0     1
  Columns 40 through 52
     0     0     0     0     0     0     0     1     0     0     0     0     0
  Columns 53 through 65
     0     1     1     0     0     1     1     0     1     0     0     1     0
  Columns 66 through 78
     0     1     0     0     0     0     1     0     0     1     0     0     1
  Columns 79 through 91
     1     0     0     0     1     0     0     0     0     1     0     0     1
  Columns 92 through 104
     0     0     1     0     1     0     0     0     1     0     0     0     1
  Columns 105 through 117
     1     0     1     0     0     1     1     1     1     1     0     0     0
  Columns 118 through 130
     0     0     0     0     0     0     0     1     1     1     1     1     0
  Columns 131 through 143
     0     0     0     0     1     0     1     0     1     1     0     0     0
  Column 144
     0
```

The FCS appended to the end of the frame by the TNC were:

```
>> Fields.FCS
ans =
  Columns 1 through 13
     0     1     0     1     0     1     1     0     0     1     0     1     1
  Columns 14 through 16
     1     1     1
```

The frame bits were used as input to the MATLAB CRC algorithm. The FCS calculated by the MATLAB CRC algorithm was:

```
>> CRC_CCITT_Generator( BitsForFCSCheck )
ans =
  Columns 1 through 13
     0     1     0     1     0     1     1     0     0     1     0     1     1
  Columns 14 through 16
     1     1     1
```

The FCS appended by the TNC and the FCS produced by the MATLAB algorithm match. Both operated independently on the same input frame bits.

5.2 Simple Test Values with Intermediate Values

The input bits for this test represented the ASCII values of “ABC” with the least-significant bit (LSB) first in the order sent.

Character	ASCII Value (Decimal)	Bit Sequence (LSB on Left)
A	65	1 0 0 0 0 0 1 0
B	66	0 1 0 0 0 0 1 0
C	67	1 1 0 0 0 0 1 0

The 24 input bits used for the test were:

```
1  0  0  0  0  0  1  0  0  1  0  0  0  0  1  0  1  1  0  0  0  0  1  0
```
(First bit sent is on left.)

The intermediate states of the shift register are (stage 1 of shift register is in left column):

```
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1   INITIAL STATE
0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1   STATE AFTER FIRST BIT
1  0  1  1  1  0  1  1  1  1  1  1  0  1  1  1
1  1  0  1  1  0  0  1  1  1  1  1  0  0  1  1
1  1  1  0  1  0  0  0  1  1  1  1  0  0  0  1
1  1  1  1  0  0  0  0  0  1  1  1  0  0  0  0
0  1  1  1  1  0  0  0  0  0  1  1  1  0  0  0
1  0  1  1  1  0  0  0  0  0  0  1  0  1  0  0
0  1  0  1  1  1  0  0  0  0  0  0  1  0  1  0
0  0  1  0  1  1  1  0  0  0  0  0  0  1  0  1
```

```
0  0  0  1  0  1  1  1  0  0  0  0  0  0  1  0
0  0  0  0  1  0  1  1  1  0  0  0  0  0  0  1
1  0  0  0  0  0  0  1  1  1  0  0  1  0  0  0
0  1  0  0  0  0  0  0  1  1  1  0  0  1  0  0
0  0  1  0  0  0  0  0  0  1  1  1  0  0  1  0
1  0  0  1  0  1  0  0  0  0  1  1  0  0  0  1
1  1  0  0  1  1  1  0  0  0  0  1  0  0  0  0
1  1  1  0  0  0  1  1  0  0  0  0  0  0  0  0
1  1  1  1  0  1  0  1  1  0  0  0  1  0  0  0
0  1  1  1  1  0  1  0  1  1  0  0  0  1  0  0
0  0  1  1  1  1  0  1  0  1  1  0  0  0  1  0
0  0  0  1  1  1  1  0  1  0  1  1  0  0  0  1
1  0  0  0  1  0  1  1  0  1  0  1  0  0  0  0
1  1  0  0  0  0  0  1  1  0  1  0  0  0  0  0
0  1  1  0  0  0  0  0  1  1  0  1  0  0  0  0  FINAL STATE (AFTER 24TH BIT)
```

The FCS is the bitwise inversion of the final state. The FCS is:

```
1  1  1  1  0  1  0  0  1  1  1  1  1  0  0  1
```

The MSB of the FCS is on the left; the MSB is sent first for AX.25 protocol (see [1]).

5.3 A Common Test Value and Bit-order Reversal

A commonly used test sequence contains the ASCII values representing the string "123456789" with the least-significant bit (LSB) of the first character sent first. The test bit sequence is (first bit in upper left with bits read left to right in each row):

```
1  0  0  0  1  1  0  0  0  1  0  0  1  1  0  0  1  1  0  0  1  1  0  0
0  0  1  0  1  1  0  0  1  0  1  0  1  1  0  0  0  1  1  0  1  1  0  0
1  1  1  0  1  1  0  0  0  0  0  1  1  1  0  0  1  0  0  1  1  1  0  0
```

Using the above bits as an input, the output of the MATLAB CRC algorithm is:

```
0  1  1  1  0  1  1  0  0  0  0  0  1  0  0  1
```

Some references report the CRC-CCITT result of "123456789" to be the hex value 0x906E (see [2]). This hex value corresponds to the binary value:

```
1  0  0  1  0  0  0  0  0  1  1  0  1  1  1  0
```

This is equivalent to the result of the MATLAB CRC algorithm with the bit order reversed. When reversed in bit order, 0x906E corresponds to:

```
0  1  1  1  0  1  1  0  0  0  0  0  1  0  0  1
```

The MATLAB CRC algorithm produces the first bit sent on the left, just like the amateur radio TNC. The output reported by [2] contains the first bit sent on the right compared to the output of the amateur radio TNC.

6 References and Links

[1] *AX.25 Amateur Packet-Radio Link-Layer Protocol*, Version 2.0, October 1984. See http://www.tapr.org/pub_ax25.html

[2] http://users.physik.tu-muenchen.de/gammel/matpack/html/LibDoc/Crypto/MpCRC.html

This information is based on an amateur packet radio project by Bill Newhall and Ileana Carrasquillo in 2008.

Source file: CRC_for_AX25_Rev_2008-11-23.doc
Revision: 11/23/2008