Error detection using CRC

The goal of this task is to become familiar with functional principles of CRC (Cyclic Redundancy Check):

- math background
- error detection features
- practical implementation

CRC is a redundant data evaluated from original information data (that we want to make secure). CRC is sent together with original data and evaluated again at the receiver site. If the newly evaluated and received CRCs differ, there is an error in the received data. If both CRCs are the same, we expect the data were received correctly. Nevertheless it exists small but non-zero probability, that the received data are incorrect.
Transmitted data bit sequence can be interpreted as a polynomial with binary coefficients. Binary sequence 11000101 can thus be represented by a binary polynomial $x^7 \oplus x^6 \oplus x^2 \oplus 1$. CRC is evaluated as an reminder after the division of the data polynomial by so called generating polynomial. During the evaluation modulo 2 addition is applied ($1 \oplus 1 = 0$).
CRC definition by the generating polynomial is ambiguous – different byte order or additional bit sequences are added before or after the data:

- sometimes bit 1 is added before data
- sometimes the sequence of N zero bits (N is degree of the generating polynomial) is appended after the data sequence and CRC is evaluated. At the receiver side the CRC evaluated from the sequence consisting of received data + CRC is then always zero (no errors are expected).

The number mentioned after the CRC letters usually specifies generating polynomial degree, e.g. CRC5 means that generating polynomial degree is 5 ($x^5 + \ldots +1$) and maximum degree of the remainder polynomial is then 4 (representing sequence of 5 bits).

1. Learn the CRC principle from the attached material.
2. Open the Matlab and the simulation schema in Fig. 1 (Plocha/crc/crc1.mdl)
   - Define the data frame with length 8 – 12 bits and store it in block *Frame,*
   - choose simple CRC polynomial (e.g. CRC-3, CRC-4) and store it in *CRC Generator* and *CRC Syndrome Detector* blocks,
   - define variables *frame_length* and *CRC_degree* in the Matlab workspace and assign them values respecting the data frame length and generating polynomial degree,
   - evaluate CRC of data frame using selected generating polynomial,
   - *Error vector* block is used to generate errors. First try no error transmission, i.e. 000……0. Error vector length is defined as a sum of data frame length and generating polynomial degree (Attention – *do not exchange degree of polynomial and its length*),
   - check the correct function of CRC using presentation blocks.
3. Using simulation check the following CRC functionality (all relevant error vectors, where it is possible):
   - if the error vector is linear shift of generating polynomial (g(x) multiplied by x^n, n = 0, 1, …), error is not detected

- more generally; if the error vector is divisible by g(x) without reminder, error is not detected
- each single bit error is detected, if generating polynomial $g(x) = x^n \oplus 1$, $n = 1, 2, \dots$
- if generating polynomial g(x) is divisible by $(x \oplus 1)$ without reminder, each error vector with odd number of bit errors is detected
- if all the polynomials $p(x) = x^i \oplus 1$, $i = 1, 2, \dots n-1$, n is a length of codeword, are not divisible by generating polynomial g(x), all double bit errors are detected
- if the error vector $e(x) = x^j(x^t \oplus \dots \oplus 1)$, $t <$ degree of g (x) – block error of length t, such error is always detected

4. Try to prove the hypothesis above as well as other two following:
   - if the error vector is the block error of length n: $e(x) = x^j(x^t \oplus \dots \oplus 1)$, $t = n =$ degree of g (x), the probability that such an error will not be detected is $2^{-(n-1)}$
   - if the error vector is the block error of length higner than n, n = degree of g (x), the probability that such an error will not be detected is 2^{-n}
5. Prepare the data transmission schema according to Figure 2 in Simulink. *Bernoulli Binary Generator* generates random binary sequence. *Binary Symmetric Channel* generates bit errors with predefined bit error probability (use e.g. 0.1‰, 1‰ a 1%). Find the probability of random event, that the error in codeword will not be detected (there will be received erroneous frame with correct CRC check).
6. Try to estimate the same probability of the same channel using the above mentioned rules. Compare the estimation with measurements.

CRC Evaluation

Data: 10011101 represented as $x^7 \oplus x^4 \oplus x^3 \oplus x^2 \oplus 1$

Generating polynomial: $g(x) = x^3 \oplus x \oplus 1$ (CRC3)

First the data are shifted left by the g(x) degree (multiplied by x^3: $x^7 \rightarrow x^{10}$ …). Then the data polynomial is divided by generating polynomial g(x). The reminder is the evaluated CRC.

$$
\begin{array}{l}
x^{10} \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 : x^3 \oplus x \oplus 1 = x^7 \oplus x^5 \oplus 1 \\
x^{10} \oplus x^8 \oplus x^7 \\
\quad x^8 \oplus x^6 \oplus x^5 \oplus x^3 \\
\quad x^8 \oplus x^6 \oplus x^5 \\
\qquad\qquad x^3 \\
\qquad\qquad x^3 \oplus x \oplus 1 \\
\qquad\qquad\quad \underline{x \oplus 1}
\end{array}
$$

CRC length equals to the degree of generating polynomial. In our case the reminder after division is $0 * x^2 \oplus 1 * x \oplus 1$, resulting CRC is then **011**.

Transmitted bit sequence is then 10011101011, represented by $x^{10} \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x \oplus 1$

CRC checking is shown below:

$$
\begin{array}{l}
x^{10} \oplus x^7 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x \oplus 1 : x^3 \oplus x \oplus 1 = x^7 \oplus x^5 \oplus 1 \\
x^{10} \oplus x^8 \oplus x^7 \\
\quad x^8 \oplus x^6 \oplus x^5 \oplus x^3 \oplus x \oplus 1 \\
\quad x^8 \oplus x^6 \oplus x^5 \\
\qquad\qquad x^3 \oplus x \oplus 1 \\
\qquad\qquad x^3 \oplus x \oplus 1 \\
\qquad\qquad\qquad \underline{0}
\end{array}
$$

Reminder is 0, no error is detected.

Name	Polynomial
CRC-1	$x + 1$ (most hardware; also known as parity bit)
CRC-3	$x^{3} + x + 1$
CRC-4-ITU	$x^{4} + x + 1$ (ITU-T G.704, p. 12)
CRC-5-EPC	$x^{5} + x^{3} + 1$ (Gen 2 RFID[15])
CRC-5-ITU	$x^{5} + x^{4} + x^{2} + 1$ (ITU-T G.704, p. 9)
CRC-5-USB	$x^{5} + x^{2} + 1$ (USB token packets)
CRC-6-ITU	$x^{6} + x + 1$ (ITU-T G.704, p. 3)
CRC-7	$x^{7} + x^{3} + 1$ (telecom systems, ITU-T G.707, ITU-T G.832, MMC, SD)
CRC-8-CCITT	$x^{8} + x^{2} + x + 1$ (ATM HEC), ISDN Header Error Control and Cell Delineation ITU-T I.432.1 (02/99)
CRC-8-Dallas/Max^im	$x^{8} + x^{5} + x^{4} + 1$ (1-Wire bus)
CRC-8	$x^{8} + x^{7} + x^{6} + x^{4} + x^{2} + 1$
CRC-8-SAE J1850	$x^{8} + x^{4} + x^{3} + x^{2} + 1$
CRC-8-WCDMA	$x^{8} + x^{7} + x^{4} + x^{3} + x + 1$[16]
CRC-10	$x^{10} + x^{9} + x^{5} + x^{4} + x + 1$ (ATM; ITU-T I.610)
CRC-11	$x^{11} + x^{9} + x^{8} + x^{7} + x^{2} + 1$ (FlexRay[17])
CRC-12	$x^{12} + x^{11} + x^{3} + x^{2} + x + 1$ (telecom systems[18][19])
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$
CRC-16-IBM	$x^{16} + x^{15} + x^{2} + 1$ (Bisync, Modbus, USB, ANSI X3.28, many others; also known as CRC-16 and CRC-16-ANSI)
CRC-16-CCITT	$x^{16} + x^{12} + x^{5} + 1$ (X.25, HDLC, XMODEM, Bluetooth, SD, many others; known as CRC-CCITT)
CRC-16-T10-DIF	$x^{16} + x^{15} + x^{11} + x^{9} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (SCSI DIF)
CRC-16-DNP	$x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^{8} + x^{6} + x^{5} + x^{2} + 1$ (DNP, IEC 870, M-Bus)
CRC-16-DECT	$x^{16} + x^{10} + x^{8} + x^{7} + x^{3} + 1$ (cordless telephones)[21]
CRC-24	$x^{24} + x^{22} + x^{20} + x^{19} + x^{18} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^{8} + x^{7} + x^{6} + x^{3} + x + 1$ (FlexRay[17])
CRC-24-Radix^-64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^{7} + x^{6} + x^{5} + x^{4} + x^{3} + x + 1$ (OpenPGP)
CRC-30	$x^{30} + x^{29} + x^{21} + x^{20} + x^{15} + x^{13} + x^{12} + x^{11} + x^{8} + x^{7} + x^{6} + x^{2} + x + 1$ (CDMA)
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$ (V.42, Ethernet, MPEG-2, PNG[22], POSIX cksum)
CRC-32C (Castagnoli)	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + 1$ (iSCSI & SCTP, G.hn payload, SSE4.2)
CRC-32K (Koopman)	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^{7} + x^{6} + x^{4} + x^{2} + x + 1$
CRC-32Q	$x^{32} + x^{31} + x^{24} + x^{22} + x^{16} + x^{14} + x^{8} + x^{7} + x^{5} + x^{3} + x + 1$ (aviation; AIXM[23])
CRC-64-ISO	$x^{64} + x^{4} + x^{3} + x + 1$ (HDLC — ISO 3309, Swiss-Prot/TrEMBL; considered weak for hashing[24])
CRC-64-ECMA-182	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^{9} + x^{7} + x^{4} + x + 1$ (as described in ECMA-182 p. 51)

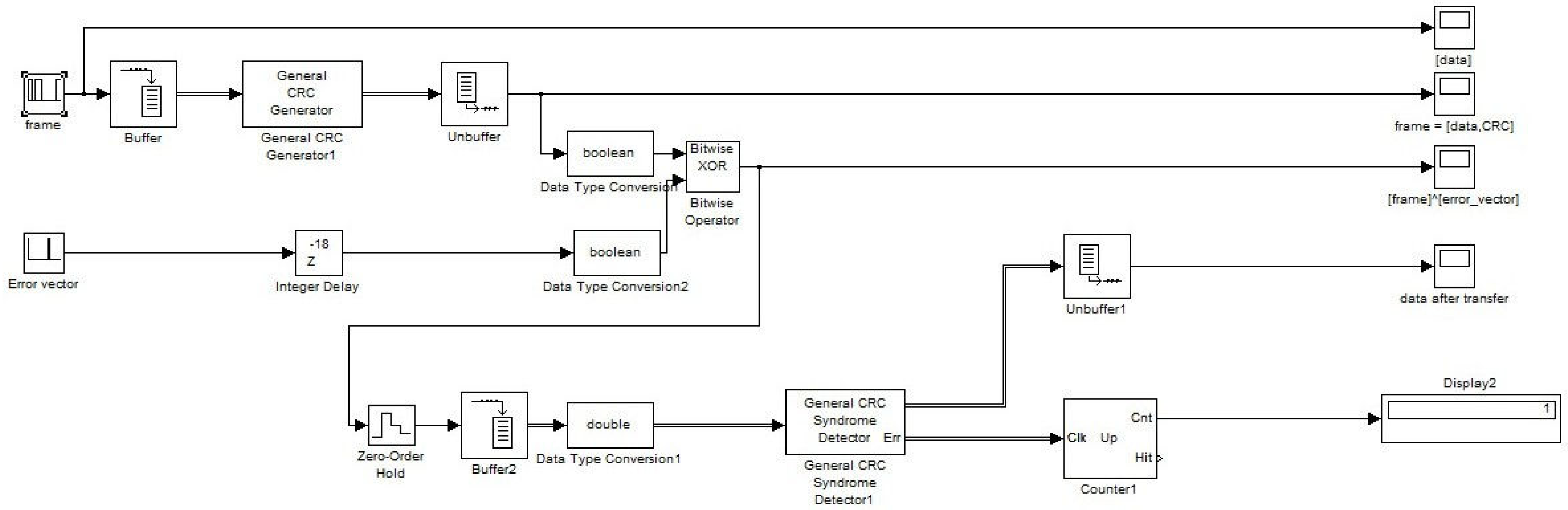


Figure 1. CRC evaluation simulation

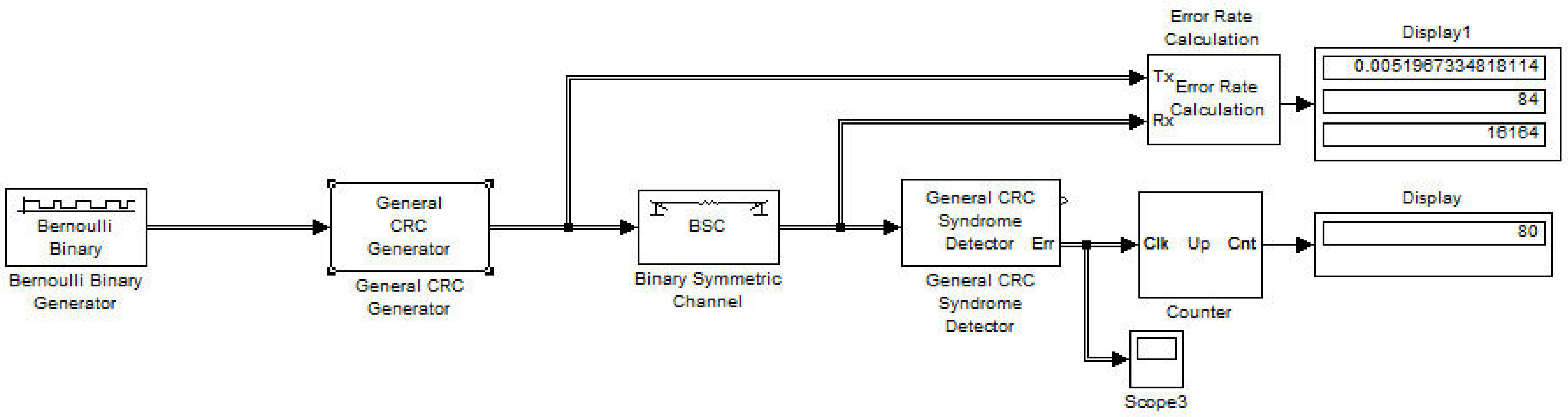
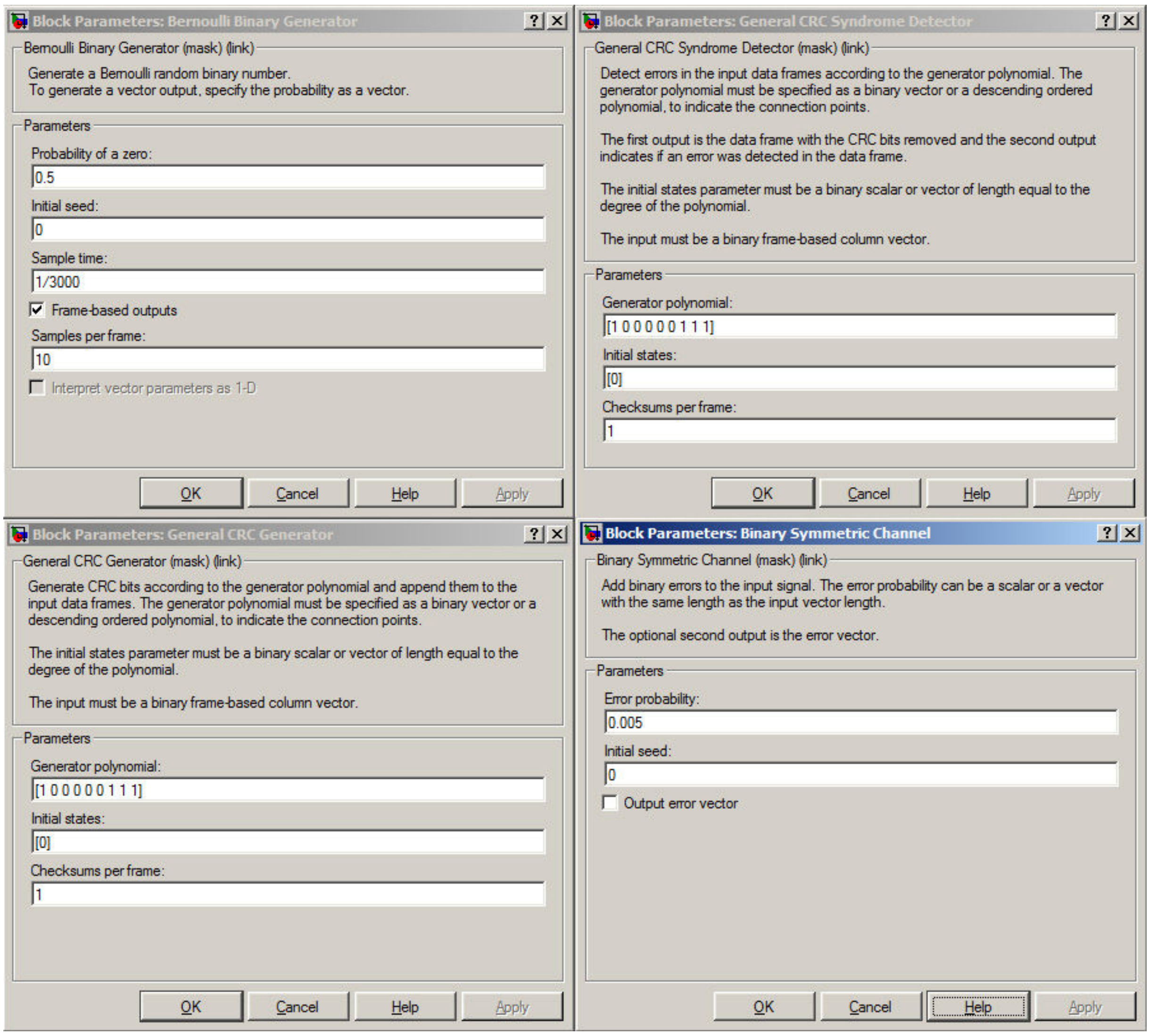


Figure 2. Simulation of random errors CRC check evaluation


Block Parameters: Bernoulli Binary Generator

Bernoulli Binary Generator (mask) (link)

Generate a Bernoulli random binary number.
To generate a vector output, specify the probability as a vector.

Parameters

Probability of a zero: 0.5

Initial seed: 0

Sample time: 1/3000

[✓] Frame-based outputs

Samples per frame: 10

[ ] Interpret vector parameters as 1-D

Block Parameters: General CRC Syndrome Detector

General CRC Syndrome Detector (mask) (link)

Detect errors in the input data frames according to the generator polynomial. The generator polynomial must be specified as a binary vector or a descending ordered polynomial, to indicate the connection points.

The first output is the data frame with the CRC bits removed and the second output indicates if an error was detected in the data frame.

The initial states parameter must be a binary scalar or vector of length equal to the degree of the polynomial.

The input must be a binary frame-based column vector.

Parameters

Generator polynomial: [1 0 0 0 0 0 1 1 1]

Initial states: [0]

Checksums per frame: 1

Block Parameters: General CRC Generator

General CRC Generator (mask) (link)

Generate CRC bits according to the generator polynomial and append them to the input data frames. The generator polynomial must be specified as a binary vector or a descending ordered polynomial, to indicate the connection points.

The initial states parameter must be a binary scalar or vector of length equal to the degree of the polynomial.

The input must be a binary frame-based column vector.

Parameters

Generator polynomial: [1 0 0 0 0 0 1 1 1]

Initial states: [0]

Checksums per frame: 1

Block Parameters: Binary Symmetric Channel

Binary Symmetric Channel (mask) (link)

Add binary errors to the input signal. The error probability can be a scalar or a vector with the same length as the input vector length.

The optional second output is the error vector.

Parameters

Error probability: 0.005

Initial seed: 0

[ ] Output error vector

Figure 3. Typical block parameters setting (simulation schema in Figure 2)

Recommended values:

Bernoulli Binary Generator:

- Probability of zero – 0.5 (the same probability for 0 and 1)
- Initial seed – e.g. 0
- Sample time → number of samples per second, set (together with simulation time) so that large enough number of frames is transmitted, e.g. 100 000.
- Samples per frame – number of bits per data frame, choose from 8 to 30

Binary Symmetric Chanel:

- Error probability – probability of bit error, try from 0.0001 to 0.01

CRC Generator:

- Generator polynomial – the same polynomial should be defined in CRC Detector
- Checksums per frame – should be 1