Formal Verification of the CRC Algorithm Properties*

P. Hlávka[1], V. Řehák[2], A. Smrčka[1], P. Šimeček[2], D. Šafránek[2], and T. Vojnar[1]

[1] FIT BUT, Brno, Czech Republic
{xhlavk00,smrcka,vojnar}@fit.vutbr.cz
[2] FI MU, Brno, Czech Republic
{xrehak,xsafran1,xsimece1}@fi.muni.cz

Abstract. This paper presents the verification of CRC algorithm properties. We examine a way of verifying of a CRC algorithm using exhaustive state space exploration by model checking method. The CRC algorithm is used for calculation of a message hash value and we focus on verification of the property of finding minimal Hamming distance between two messages having the same hash value. We deal with 16, 32 and 64 bits CRC generator polynomials, especially with one used in the Liberouter project.

1 Introduction

The FlowMon is one of network adaptors developed within the Liberouter—Programmable Hardware [8] project, which aim is to create a monitoring probe for gathering a summary information of network data flows. The FlowMon [6] keeps data of current opened flows and identifies the flow for every incoming packet via 64-bit CRC hash value. One of the common properties of all hash functions is that, if input message has greater length than the length of the hash value, it produces *collisions* (or there exist *colliding messages*)—different input messages with the same hash value. These collisions are undesirable when using hash algorithm for message identification. Our goal is to get the minimal Hamming distance of two colliding messages.

Owing to the fact that the CRC algorithm has strong mathematical basis, most of the effort is given to verifying of the selected generator polynomial, which is the base of the CRC functionality. One of interesting properties is minimal Hamming distance of any two colliding messages. This information is obtained by the CRC algorithm simulation or by formal verification. Exhaustive verification of the long-length polynomials (high order polynomials) is impossible due to the state space explosion problem and therefor the simulation approach offers more satisfying results. But exhaustive state space exploration has also some interesting results for the length of 32 bit CRC algorithms [4] (use of the cluster computer for verifying the properties of CRC32-IEEE generator polynomials). Further information on a similar topic can be found in [2] (special algorithm for calculating minimal Hamming distance of two colliding messages) and [5] (finding the most suitable generator polynomial).

We will discuss an unusual way of CRC verification, i.e., we use the model checking method for exhaustive state space exploration and formal verification of the minimal

* This research has been supported by the CESNET activity “Programmable hardware” [8].

Hamming distance. At first, we will describe some main properties of the CRC algorithm, the second chapter deals with searching collisions. The next chapter describes our implemented model for the algorithm of finding colliding messages and minimal Hamming distance. We have also realised some experiments in three non-commercially model checkers (Cadence SMV [3, 9], NuSMV [1] and Spin [7]). The last chapter discusses the most important results of our experiments.

2 Cyclic Redundancy Check

The Cyclic Redundancy Check (CRC) algorithm was originally developed for the detection of line transmission errors. It is designed to be fast and easy to implement in a hardware by using logic *XOR* (exclusive OR) gates and shifters. The algorithm provides very good protection from burst errors, which are typical for transmission lines. Thanks to easy implementation, it is useful for error detection but cannot safely rely on data integrity verification.

Mathematically, the CRC is based on division operation over Galois Field $GF(p^n)$. If $n > 1$, the elements of the $GF(p^n)$ can be represented as polynomials which coefficients belong to $GF(p)$ and the maximum order of polynomials is bounded from above by n. If $p = 2$, as for the CRC algorithm, coefficients of $GF(2)$ are 0 or 1 and the polynomial can be written as a binary vector.

Ordinary arithmetical operations such as addition, subtraction, multiplication, and division closed on $GF(2^n)$ are depicted in Figure 1.

Polynomial as a binary vector in $GF(2^n)$

$$(x^6+x^5+x^2+x+1) = (01100111)_2 = (67)_{16}$$

$$(x^7+x^6+x^5+x^4+1) = (11110001)_2 = (\mathrm{F1})_{16}$$

Addition $(1111+101=1010)$

$$(x^3+x^2+x+1)+(x^2+1) = x^3+2x^2+x+2 = x^3+x$$

Subtraction $(1011-101=1110)$

$$(x^3+x+1)-(x^2+1) = x^3-x^2+x+2 = x^3+x^2+x$$

Multiplication without modulo reduction $(101 \cdot 11 = 1010)$

$$(x^2+x)(x+1) = x^3+2x^2+x = x^3+x$$

Division with remainder $(1011 = 101 \cdot 11+1)$

$$x^3+x+1 = (x^2+x)(x+1)+1$$

Fig. 1. Examples of arithmetical operations over $GF(2^n)$

The result of the CRC algorithm, a so-called *hash value* (commonly also known as a *checksum*), is defined as the remainder after a division of input message by a gen-

erator polynomial over $GF(2^n)$. The generator polynomial is the main CRC algorithm parameter which influences error detection capabilities or hash quality. The bit length of a hash value (reminder after a division) depends on the order of generator polynomial (divisor). If the order of a divisor is n then the order of a reminder is at most $n-1$. Therefore, the length of a reminder binary representation (length of a hash value) is n.

In a real application, the hash value is computed from the input message extended with n zero bits appended to the end of an input message and transmitted. An input message together with its hash value produce a message which is divisible by the generator polynomial. The side, which receives the message with the appended hash value, computes a checksum from the whole incoming message. The incoming message is correct, if the calculated hash value is a zero-valued vector.

3 Hash Collisions

If the length of the incoming message (hash function input) is greater than the length of the produced hash value then there always exist at least two distinct input messages with the same hash value. Such input messages are called *hash collisions* (or *colliding messages*). We focused mainly on the minimal number of bits for which the two colliding messages are different (Hamming distance of colliding messages) and partly on the position of such bits.

Minimal Hamming distance is one of the most important parameters for error detection codes but there is no easy way how to figure it out from an arbitrary CRC generator polynomial. However, there are rules for deciding whether the chosen generator polynomial is capable of n-bit errors (changes) detection for some selected n.

Let:

- $T(x)$ be a polynomial (binary vector) representing an *input* (*incoming*) message of order $m-1$ (length of an input vector is $|T(x)| = m$ bits).
- $E(x)$ be a polynomial (binary vector) of the order $m-1$ ($|E(x)| = |T(x)| = m$ bits). Each vector coefficient of $E(x)$ with value 1 corresponds to the input message $T(x)$ coefficient that has been inverted. In a real application, this polynomial represents an errors during message transmission. Therefore, we call this vector as the *error vector*
- $G(x)$ be a generator polynomial of the order n ($|G(x)| = n+1$ bits).

There is no need to show here the whole complex mathematics of CRCs (see [10] for further details). For our purposes, the following simplified equations will be sufficient. Note that the arithmetical operations in polynomials expression, such as '$+$' or '**mod**', are closed in $GF(2^n)$.

The collision appears when two input messages $T_1(x) \neq T_2(x)$ have the same hash value:

$$T_1(x) \textbf{ mod } G(x) = T_2(x) \textbf{ mod } G(x) \tag{1}$$

These two vectors differ in several bits kept in $E(x)$ polynomial.

$$E(x) = T_1(x) + T_2(x) = T_2(x) + T_1(x) \tag{2}$$

Let us assume that

$$T_2(x) = T_1(x) + E(x) \tag{3}$$

is a colliding message to the original message $T_1(x)$. From (1) and (2) we get:

$$T_1(x) \textbf{ mod } G(x) = (T_1(x) + E(x)) \textbf{ mod } G(x) \tag{4}$$

$$T_1(x) \textbf{ mod } G(x) = T_1(x) \textbf{ mod } G(x) + E(x) \textbf{ mod } G(x) \tag{5}$$

$$\text{which holds if} \quad E(x) \textbf{ mod } G(x) = 0 \tag{6}$$

Such an $E(x)$ is called *colliding error vector* (*colliding error polynomial*).

The CRC algorithm is said to be a non-secure hash function because there is an easy way how to generate any collisions to arbitrary input message. This could be done by creating a colliding error polynomial in the following manner:

$$E(x) = x^j G(x), \quad \text{where } |E(x)| = m,\ |G(x)| = n+1,\ j \in \langle 0; m-n-1 \rangle, \tag{7}$$

then all input $T(x) + E(x)$ have the same hash as the original input message $T(x)$ because the error polynomial $E(x)$ is divisible by the generator polynomial $G(x)$. More complex error patterns could be created using the colliding error polynomial

$$E(x) = x^{j_i} G(x) + x^{j_{i-1}} G(x) + \cdots + x^{j_0} G(x) \tag{8}$$

which is divisible by the generator polynomial $G(x)$ for arbitrary $i > 0$ and $j_i \in \langle 0; m-n)$. To generate unique collision error polynomials, only elements when $(i < m-n) \wedge (j_k = j_m \Rightarrow k = m)$ are meaningful. See figure 2 for the simple example of a collision error polynomial.

Proposition 1. *There are* $2^{m-n} - 1$ *distinct non-zero collision error polynomials* $E(x)$ *for the input message length* m *and the hash length* $n+1$.

Proof. If the input message length is m (the order of $T(x)$ is $m-1$) and the length of generator vector is $n+1$ (the order of $G(x)$ is n). There exists $x^j, j \in \langle 0; m-n)$ such that the order of a product $x^j G(x)$ is less than m ($x^{j_{max}} x^n = x^{m-n-1} x^n = x^{m-1}$). According to (8), the error polynomials are created as the sum closed in $GF(2)$ of various combination of products $x^j G(x)$. Number of such combinations is 2^{m-n}. There are no two combinations producing equal error vector, because the equation

$$\begin{aligned} x^{j_p} G(x) + x^{j_{p-1}} G(x) + \cdots + x^{j_0} G(x) &= x^{k_r} G(x) + x^{k_{r-1}} G(x) + \cdots + x^{k_0} G(x) \\ x^{j_p} + x^{j_{p-1}} + \cdots + x^{j_0} &= x^{k_r} + x^{k_{r-1}} + \cdots + x^{k_0} \end{aligned} \tag{9}$$

holds only if $p = r \wedge \forall i \in \langle 0; p \rangle : j_i = k_i$. There are one zero-valued vector $E(x)$ where no $G(x)$ is present, so the final number of non-zero collision error polynomials is $2^{m-n} - 1$.

4 Searching the Minimal Hamming Distance

Once we have a set of all existing collision error polynomials $E(x)$ generated from a sum of all possible combinations of products $x^j G(x)$ derived from (8), we can generate every colliding message T_2 to an input message T_1 by applying (3) as depicted in figure 2.

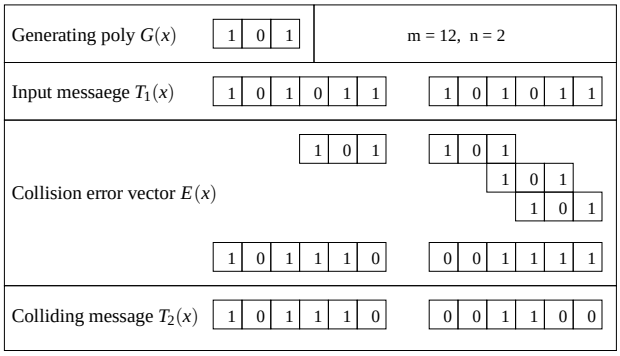

Fig. 2. An example of colliding messages differential by equation (8). Here, the collision error polynomial $E(x)$ equals a sum of the four $G(x)$ elements multiplied by x^0, x^1, x^3, and x^6.

The number of bits, in which are two colliding input messages $T_1(x)$ and $T_2(x)$ different, is reflected in the collision error polynomial $E(x)$. The minimal Hamming distance is equal to the minimal number of non-zero coefficients among all possible error polynomials.

For the implementation, within $GF(2)$, we can rewrite the equation 8 to a form:

$$E(x) = c_{m-n-1}x^{m-n-1}G(x) + c_{m-n-2}x^{m-n-2}G(x) + \cdots + c_0x^0G(x), \qquad (10)$$
$$\text{where } c_i \in \{0,1\} \text{ for } i \in \langle 0, m-n-1\rangle$$

Then, the generation of $E(x)$ is based on progressive summarising of $c_ix^iG(x)$, $i \in \langle 0, m-n-1\rangle$. The algorithm of searching the minimal Hamming distance uses such a calculation, chooses randomly the c_i coefficients, and counts the non-zero occurrences of x^i elements in currently generated $E(x)$.

Our model implements the generation of collision error polynomials using shift and xor operations (Figure 3). It contains a buffer of the same length as the generating polynomial. This buffer represents a window to currently generated $E(x)$. The symbolic representation of the algorithm is the following:

```
polynomial := CRC_generator_polynomial // vector representing G(x)
buffer := (0,0,...,0)                  // window to generated E(x)
counter := 0                           // counter of min. Hamming distance
carrybit := 0                          // carry bit after shift operation
c_i := 0                               // current c_i coefficient
for i := 0 to m-n-1 do
    c_i := choose_random(0,1)
    if c_i then
         buffer := buffer xor polynomial
    end if
    carrybit, buffer := shift_left(buffer)
    counter := counter + carrybit
end for
```

At the end of every iteration, the counter value plus the number of non-zero bits of a buffer equals to a number of non-zero bits of currently generated error message $E(x)$. Using state space search methods and the number of steps bounding, it is possible to find the sequence of shifts and nondeterministic xors of generating polynomial such that the number of non-zero bits of the error message will be minimal. We can obtain such an information by running model checking several times with different parameters—we are looking for the maximum number of non-zero bits for which the property (see Figure 3) is satisfied. This number is equal to the minimal Hamming distance for the configuration of the selected polynomial and the respective input message length. Such an approach is more effective than searching the whole input space and searching for collisions.

Owing to the state space explosion problem in the model checking algorithm, we have implemented the described model using model checkers (Cadence SMV [3, 9], NuSMV [1] and Spin [7]) and tried to figure out the minimal Hamming distances for several generating polynomial and input message lengths.

```
#define tested_HD 18
module main() {
    polynomial : array 0..64 of boolean;
    buffer : array 0..64 of boolean;
    counter : 0..tested_HD+1; rand : boolean;

    polynomial[0] := 1; polynomial[1] := 0; polynomial[2] := 0;
    polynomial[4] := 0; . . .; polynomial[64] := 1;

    rand := {0,1};

    init(counter) := 0;
    next(counter) := counter + buffer[0];

    init(buffer) := polynomial;
    if (rand) { next(buffer) := (buffer << 1) ^ polynomial; }
    else      { next(buffer) := (buffer << 1); }

    property : assert G(counter+buffer[0]+buffer[1]+...+buffer[64]
                        < tested_HD);
}
```

Fig. 3. The sketch of the verification model of CRC algorithm for Cadence SMV. This model is prepared for testing minimal Hamming distance = 18, which is a settings for the last CRC64-Liberouter polynomial verification (see Table 1).

5 Experiments

To simplify the model and to reduce space requirements, we have implemented three separate models for hash value length of 16, 32 and 64 bits. In cases when we are searching for general n-bit error detection, we are not bounding the length of the input

message. In other cases, the input message length can be bounded by adding a special counter to the model or by using bounded model checking, which does not cause any additional state space extensions.

Table 1 shows some interesting results from the verification of five different generator polynomials. The name and the CRC hash value length is expressed in the first column (e.g., CRC32-IEEE is generator polynomial used in ethernet networking for 32 bits checksum). The second and the third column is the settings for the verification. Other columns represent the results of verification (time, state space and counterexample length). If the last column has no value, the current settings of model checking fulfils the property, i.e., counted minimal Hamming distance is smaller than set value (the second column).

The result of the verification for the input message m, hash length $n+1$, and the minimal number of non-zero bits e of the error polynomial is either (i) the prove that there *is not* any collision in selected polynomial $G(x)$ between two input messages, which differ in e bits, (ii) or the counterexample of the length p which shows the collision for the input message length $p+n+1$.

The space requirements of the verification depends exponentially on the length of the input message and also on the length of the hash value. For each of the selected hash lengths we have tried to figure out the maximal input message length, which can be successfully verified. Then, on average input message lengths, we have tried to compare two selected polynomials for each hash length.

6 Conclusion

In this paper, we have discussed the model checking approach of exhaustive verification of the CRC algorithm property. We have introduced a basis of the CRC algorithm and its weakness in the field of data integrity. The main problem still remains in searching of minimal Hamming distance of colliding messages for the CRC with generator polynomials with order greater or equal 64. The proposed approach has been implemented in NuSMV, Spin and Cadence SMV model checkers (Cadence SMV gives the best results). The future work can be focused on the algorithm of finding the best generator polynomial $G(x)$.

References

1. A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri. NuSMV: a new Symbolic Model Verifier. In *Proceedings of the International Conference on Computer-Aided Verification*, volume 1999, pages 495–499. Springer Verlag, 2005.
2. Peter Kazakov. Fast Calculation of the Number of Minimum-weight Words of CRC Codes. volume 47, pages 1190–1195, 2001.
3. Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. ISBN 0-7923-9380-5.
4. Philip Koopman. 32-Bit Cyclic Redundancy Codes for Internet Applications. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 459–472, Washington, DC, USA, 2002. IEEE Computer Society.

$G(x)$ polynomial	Input length [bits]	minimal Hamming distance	Checking time [s]	BDD nodes	State space	Steps of counterexample
CRC16-CCITT	1023	2	181.20	2450800	$2.09 \cdot 10^6$	-[2]
CRC16-CCITT	2047	2	830.65	5648036	$8.37 \cdot 10^6$	-[2]
CRC16-CCITT	∞	2	197.44	-[1]	196571	32752
CRC16-CCITT	∞	3	185.06	-[1]	262143	-[2]
CRC16-CCITT	∞	4	-	-	-	1
CRC16-Baicheva	127	2	1.15	-[1]	32515	-[2]
CRC16-Baicheva	1023	2	1.94	-[1]	36183	136
CRC16-Baicheva	63	4	51.16	4808185	$1.97 \cdot 10^6$	-[2]
CRC16-Baicheva	127	4	44.28	2931907	$1.06 \cdot 10^6$	-[2]
CRC16-Baicheva	255	4	43.87	3174045	$1.06 \cdot 10^6$	136
CRC32-IEEE	126[3]	4	614.33	-[1]	-[1]	-[2]
CRC32-IEEE	32	6	246.83	24866700	$3.07 \cdot 10^6$	-[2]
CRC32-IEEE	32	8	596.32	35689500	$5.08 \cdot 10^6$	-[2]
CRC32-IEEE	32	10	132.32	16285873	$2.47 \cdot 10^6$	22
CRC64-Liberouter	15	14	20.92	5750812	65535	-[2]
CRC64-Liberouter	15	16	21.13	5751059	65535	-[2]
CRC64-Liberouter	15	18	2.19	376830	8191	13
CRC64-ECMA	15	24	33.25	5750915	65535	-[2]
CRC64-ECMA	15	26	9.36	1762147	16383	14

[1] – Could not be figured for chosen verification tool (NuSMV)

[2] – No counterexample found, there is not any error vector $E(x)$ with chosen parameters.

[3] – Bounded model checking was used.

Table 1. Examples of results for selected polynomials. Most of them are from the Cadence SMV model checker

5. Philip Koopman and Tridib Chakravarty. Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, page 145, Washington, DC, USA, 2004. IEEE Computer Society.
6. J. Kořenek and T. Pečenka M. Žadník. NetFlow Probe Intended for High-Speed Networks. In *Proceedings of the 15th International Conference on Field-Programmable Logic and Applications*, pages 695–698. IEEE Computer Society, 2005.
7. Bell Labs. *SPIN Model Checker*. http://spinroot.com/spin/whatispin.html.
8. Liberouter Project Homepage. http://www.liberouter.org/.
9. Ken McMillan. *Cadence SMV Manual*, 2006. http://www.cis.ksu.edu/santos/smv-doc/.
10. Andrew Tanenbaum. *Computer Networks (fourth edition)*. Prentice Hall, Upper Saddle River, NJ, 2003. ISBN 0-13-038488-7.