Towards Modeling and Verification of Eurobalise Telegram Encoding Algorithm

Sergey Staroletov

Polzunov Altai State Technical University

serg_soft@mail.ru

November 18, 2022

Eurobalise – definition

Eurobalise

Eurobalise is a transmitter that broadcasts data on the current state of the railway to a passing train.

- It is a crucial part of the European Train Control System, as well as other automatic trains protection systems.
- Such a transmitter is activated with an approaching train and dispatches encoded messages called telegrams.

The need for standards

- Complex systems are hard to understand.
- For real-world (industrial) systems, their behaviour are specified in standards with some degree of formality.
- The standards should be the main source for further system formalization and checking[1].

[1] The approach has already tried by using European standards of brake safety to test an ABS model: Staroletov, S. (2021, May). Modeling the Anti-Lock Braking System in Scilab and Its Checking for Compliance with Uniform Requirements. In International Conference on Industrial Engineering (pp. 413-424). LNME. Springer, Cham.

Eurobalise – open standard

The standard

Encoding/decoding rules for Eurobalise telegrams are well described in the freely available standard[2]

ALSTOM * ANSALDO * BOMBARDIER * INVENSYS * SIEMENS * THALES

ERTMS/ETCS – Class 1

FFFIS for Eurobalise

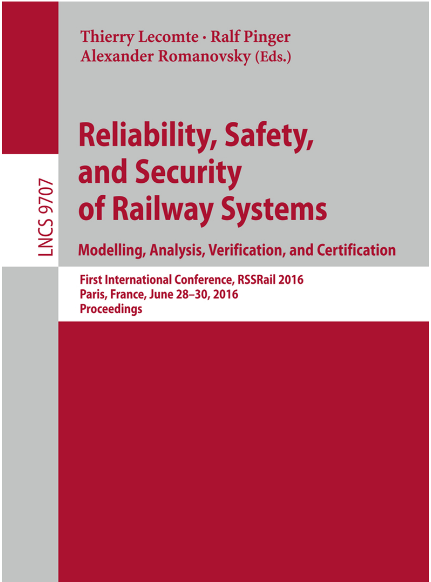
REF :	SUBSET-036
ISSUE :	2.4.1
DATE :	September 27, 2007

[2] https://www.era.europa.eu/sites/default/files/filesystem/ertms/ccs_tsi_annex_a_-_mandatory_specifications/set_of_specifications_1_etcs_b2_gsm-r_b1/index009_-_subset-036_v241.pdf

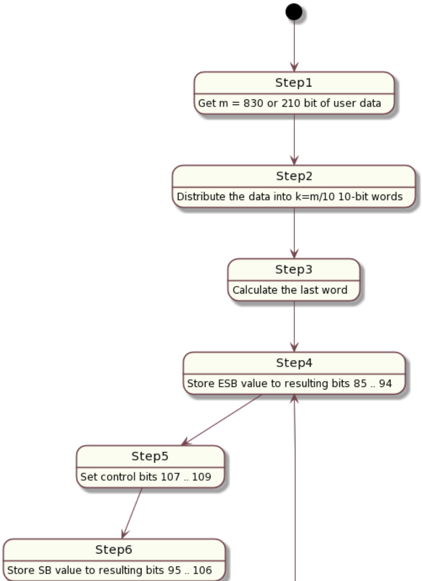
Task setting

- In this study, we consider the encoding process of telegrams that are broadcasting by eurobalises.
- This process is implemented in accordance with the open international standard adopted by Alstom, Ansaldo, Bombardier, Siemens and Invensys.
- Our motivation is to apply formal methods as well as non-deterministic programming to check the operation of such systems as well as create verifiable models based on uniform requirements.

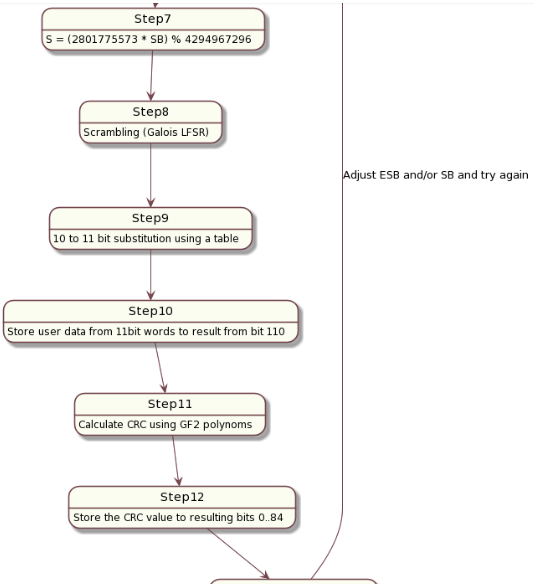
On the need of Safety



Encoding algorithm scheme – 1



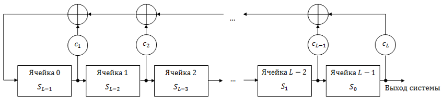
Encoding algorithm scheme – 2



Some interesting points

- Galois LFSR scrambling.
- 10 to 11 bits substitution.
- CRC calculation using GF2 polynomials.
- Adjusting ESB and SB.
- Checking the telegram candidate.

Galois LFSR scrambling




```
unsigned int S = (2801775573 * sb_state) % 4294967296;
unsigned int shift = S;
//h = 11110101000000000000000000000001 = F5000001
unsigned int toggle = 0xF5000001;
for (int i = 0; i < k; i++) //forall telegram parts
 for (int bb = 0; bb < 10; bb++) { //forall bits
   // get the next user bit
   char bit = getBit(((byte *)&U[i]), bb);
   //most significant bit
   char msb = (shift >> 31) & 1;
   //output = input xor msb(shift)
   setBit(((byte *)&U[i]), bb, (bit ^ msb));
   //shift
   shift <<= 1;
   //apply h
   if (bit) shift ^= toggle;
}
```

10 to 11 bits substitution

```
extern short words11 [] = {
00101, 00102, 00103, 00104, 00105, 00106, 00107, 00110, 00111, 00112
00113, 00114, 00115, 00116, 00117, 00120, 00121, 00122, 00123, 00124
00125, 00126, 00127, 00130, 00131, 00132, 00133, 00134, 00135, 00141
00142, 00143, 00144, 00145, 00146, 00147, 00150, 00151, 00152, 00153
00154, 00155, 00156, 00157, 00160, 00161, 00162, 00163, 00164, 00165
00166, 00167, 00170, 00171, 00172, 00173, 00174, 00175, 00176, 00201
00206, 00211, 00214, 00216, 00217, 00220, 00222, 00223, 00224, 00225
00226, 00231, 00233, 00244, 00245, 00246, 00253, 00257, 00260, 00261
 ...
}
```

Final CRC calculation using GF2 polynomials

The encoding process begins with splitting user data into 10-bit words, then using a lookup table to replace them with 11-bit words. The data is then scrambled and written to the result along with an initial state of the scrambler and some extra shaping bits. Finally, the result also stores the CRC value obtained due to the operation with polynomials over $\mathbb{F}_2$ representing user data and prefitted values (1).

$$b_{84} \cdot x^{84} + .. + b_1 \cdot x + 1 = Rem_{f(x) \cdot g(x)}[b_{n-1} \cdot x^{n-1} + .. + b_{85} \cdot x^{85}] + o(x) \quad (1)$$

Where $f(x) = f_l(x)$, $g(x) = g_l(x)$ and $o(x) = g_l(x)$ for the long format, while $f(x) = f_s(x)$, $g(x) = g_s(x)$ and $o(x) = g_s(x)$ for the short format. These polynomials can be viewed in the following binary form:

$$f_l = 11011011111 \, \mathrm{deg}(10)$$

$$g_l = 101110001000011100111001101001111010001011101101010100100011101 \, \mathrm{deg}(75)$$

$$f_s = 10110101011 \, \mathrm{deg}(10)$$

$$g_s = 1001111101111001000011000010111111101111011111100101001001010001 \, \mathrm{deg}(75)$$

Note that $g_l(x)$ and $g_s(x)$ satisfy some rule which implies that the three fold

Operations on polynomials over $\mathbb{F}_2$

- $\mathbb{F}_2$ or GF(2) is the Galois field with two elements. It comprises additive and multiplicative identities, respectively, denoted 0 and 1. Its addition is the same as the logical XOR operation; the subtraction is the same operation as the addition, while the multiplication is identical to the logical AND operation.
- Operations with polynomials in this field are similar to operations over the field $\mathbb{R}$, but in our case, all operations are performed by modulo 2. Next, it is assumed that P_1 and P_2 are polynomials over the $\mathbb{F}_2$ having given degrees, and R is the result of the operation.

Operations on polynomials over $\mathbb{F}_2$

Addition:

$$\begin{aligned} R_i = P_{1i} + P_{2i} \pmod 2, i \in (\deg(P_1)..0], \deg(P1) = \deg(P_2) \vee \\ R_i = P_{1i}, i \in (\deg(P_1).. \deg(P_2)] \wedge R_j = P_{1j} + P_{2j} \pmod 2, j \in (\deg(P_2)..0], \deg(P_1) > \mathrm{de} \\ R_i = P_{1i}, i \in (\deg(P_2).. \deg(P_1)] \wedge R_j = P_{1j} + P_{2j} \pmod 2, j \in (\deg(P_1)..0], \deg(P_1) < \mathrm{de} \\ \deg(R) = \mathit{check}(\mathit{max}(\deg(P_1), \deg(P_2))) \end{aligned} \tag{2}$$

Multiplication:

$$\begin{aligned} R_{i+j} = R_{i+j} + P_{1i} \cdot P_{2j} \pmod 2, i \in (\deg(P_1)..0], j \in (\deg(P_2)..0] \\ \deg(R) = \mathit{check}(\deg(P_1) + \deg(P_2)) \end{aligned} \tag{3}$$

check: The resulting degree should be adjusted according to the actual degree as a position of first 1 for the cases like 1+1=0.

Division: division using the column strategy

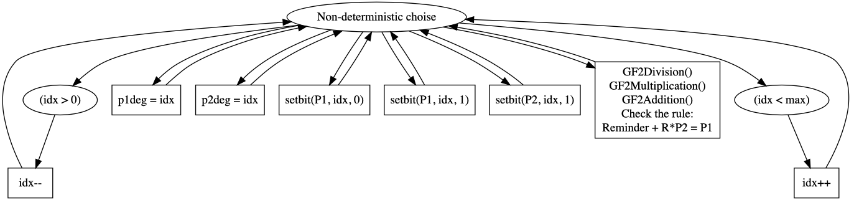
Verification of our polynomial library

When implementing a reliable library to work with $\mathbb{F}_2$ polynomials, we used the following steps:

1. We have implemented formulas as a program in the C language.
2. We tested the implementation on several tests according to gf2test[3].
3. Next, we have implemented the algorithms in **Promela**, taking into account the Promela language features. The Promela implementation is based on the original C implementation.
4. We also tested the resulting program in the modeling language according to gf2test.
5. After that, we actually proceeded to verification. We implemented a non-deterministic transition system using *od-do* operator and conditions that can be true simultaneously inside a non-deterministic *if* clause.
6. During the verification run, we got several erroneous paths of execution, which made it possible to correct the implementation errors.

[3] `http://sharetechnote.com/html/Handbook_Communication_GF2.html`

Verification of our polynomial library



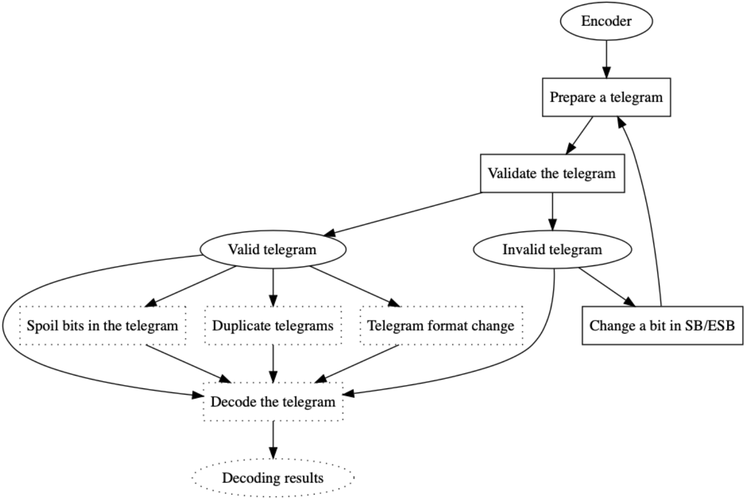
Encoding example

Test telegram encoding for the message 'Hello, IrGUPS':

```
size words= 1024
original:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000101001101010000010101010100011101110010010010010010000000101100011011110
1101100011011000110010101001000
U:
0011000111 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000101 0011010100 0001010101 0100011101
1100100100 1001001000 0000101100 0110111101 1011000110 1100011001 0101001000
12 scrambling bits
B = 2001
S = 1420600293

 After scrambling:
1011011101 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000010100 1101010000 0101010100 0001101110 0001001111 1101100010 0001101001
0111010010 0000101010 0001110001 0111111001 1100000001 1001011010 1100010010
U11:
10110111110 00001000001 00001000001 00001000001 00001000001 00001000001 00001000001 00001010101 11010000110 01010011001 00100001000 00010110001 11010100011
00100000011 01110011101 00001101110 00100001011 01111110111 11000001011 10010110000 11000100010
result telegram:
10110111110000010000010000100000100001000001000010000010000100000100001010101110100001100101001100100100001000000101100011101010001100100000110
1110011101000011011100010000101101111110111110000010111001011000011000100010 001 011111010001 0001100011
000101001000001001101110010111111111101111000111110100000110010111011001011001101001000
```

Our verification strategy



Implementation of the Eurobalise encoding procedure in the model language for verification purposes

Promela

Promela – a C-like actor-based model language for SPIN verifier

- A CSP-followed language inspired by Clarke's EMC.
- Formal semantics.
- Support a lot of arithmetic[4].

[4] Staroletov, S., Shilov, N. (2019, July). Applying model checking approach with floating point arithmetic for verification of air collision avoidance maneuver hybrid model. In International Symposium on Model Checking Software (pp. 193-207). Springer, Cham.

Implementation of the Eurobalise encoding procedure in the Promela model language

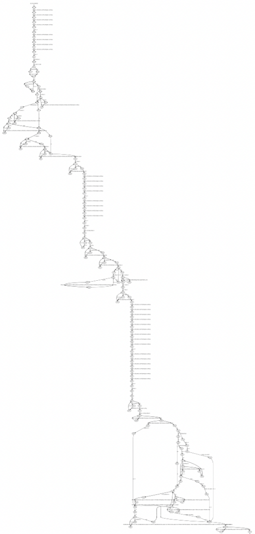
- Implementation of work with bit data in arrays is necessary (+).
- Need to work with GF(2) polynomials (+).
- We need to implement scrambling and other parts of the telegram encoding algorithm according to the specification (+).
- Optional: work with int64 and unsigned int32 (-).

Implementation of the Eurobalise encoding procedure in the Promela model language – code sample [5]

```
if
  ::(resultLen == LEN_LONG) -> {
    short data_size = 1023 - 85;
    //prepare inputs
    for (i : 0 .. 1) {
      P1[i] = fl[i];
    }
    for (i : 0 .. (75 / 8 + 1)) {
      P2[i] = gl[i];
    }
    //mult, result in R
    GF2Multiplication(fg_degree, 10, 75);
    //copy resut to P2
    for (i : 0 .. (fg_degree / 8 + 1)) {
      P2[i] = R[i];
    }
    //copy data to P1
    for (i : 0 .. (data_size / 8 + 1)) {
      P1[i] = data_to_sign[i];
    }
    //div and get the reminder
    GF2Division(div_deg, rem_deg, data_size, fg_degree, isOk);
    //copy result to P1
    for (i : 0 .. (rem_deg / 8 + 1)) {
      P1[i] = RemResult[i];
    }
    //copy gl to P2
    for (i : 0 .. (75 / 8 + 1)) {
      P2[i] = gl[i];
    }
    //add
    GF2Addition(crc_deg, rem_deg, 75);
    //result will we in R
  }
  ::else -> {//LEN_SHORT
```

[5] https://github.com/SergeyStaroletov/PromelaSamples/blob/master/Eurobalise.pml

Implementation of the Eurobalise encoding procedure in the Promela model language – obtained transition system



Verification of our polynomial library

Sketch code of the solution in Promela is presented below:

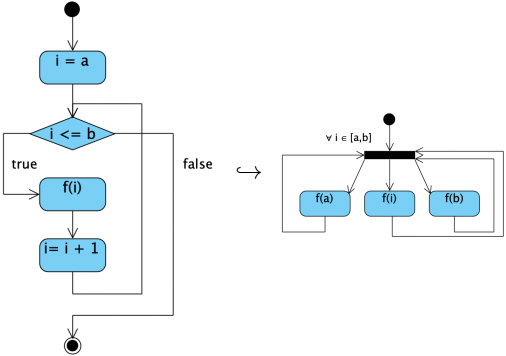
```
do
  //Change the index
  ::(idx > 0) -> idx--;
  ::(idx < max) -> idx++;
  //Change the degrees
  ::true -> p1deg = idx;
  ::true -> p2deg = idx;
  //Put 1/0 at some place in P1/P2 polynomials
  ::true -> setbit(P1, idx, 0);
  ::true -> setbit(P2, idx, 0);
  ::true -> setbit(P1, idx, 1);
  ::true -> setbit(P2, idx, 1);
  ::true -> {
      // GF2Division()
      // GF2Multiplication()
      // GF2Addition()
      // Check the rule Reminder + R*P2 = P1
  }
```

Implementation of the Eurobalise encoding procedure in the model language – requirements

- $[](C_1 \wedge C_2 \wedge C_3 \wedge C_4)$, where
- C_1 – Alphabet Condition;
- C_2 – Off-Synch-Parsing Condition;
- C_3 – Aperiodicity Condition for Long Format;
- C_4 – Under-sampling Condition (the standart, page 43).

Non-deterministic programming

Floyd used the expression ”non-deterministic algorithms”[6] to indicate those containing special commands performed by selecting one parameter from several.



[6] Floyd, Non-deterministic Algorithms, Journal of the ACM, Vol. 14, no. 4, October 1967.

Implementation of the Eurobalise encoding procedure in a model language with non-deterministic operators

What can be done non-deterministically related to the operation with Eurobalise telegrams:

- Verified library for working with polynomials (+).
- Changing the encoder parameters in case of deviation from the correctness of the telegram.

Conclusion

- In this work, we examined the use of modeling languages and non-deterministic programming for solving the problem of encoding telegrams in modern railway networks. We can note that such algorithms are well specified in the Promela language because it was created specifically for modeling data transmission protocols.
- We have learned how to verify the correctness of encoding according to the validation requirements from the standard.
- In the future, we are going to implement a model for the Eurobalise telegram decoder and verify both encoding and decoding of messages by inserting random data into telegrams, duplicating them, using short telegrams instead of long ones, and so on.