BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

RDS Encoder and Decoder

Wireless and Mobile Networks – Assignment

December 10, 2024

Michal Blažek

Contents

1 Introduction

Radio Data System (RDS) is a communications protocol standard for embedding small amounts of digital information in conventional FM radio broadcasts. It was developed by the European Broadcasting Union (EBU) Member countries with the goal to create an internationally agreed standard for such a system. The specification was initially published in 1984 and then revisited by the European Committee for Electrotechnical Standardization (CENELEC) in 1998. This project is based on the 1998 specification [1].

There are 15 message group types (resp. 30 – each type has two variants) used to transmit various information. For example Group 0 is used to transmit basic tuning and switching information, Group 2 encodes Radio Text. For ilustration imagine you are driving your car with the radio turned on. The message that you see in the radio dashboard (eg. information about the song that is currently playing) was probably transmitted to your car using the RDS Group 2.

The basic structure for messages of all group types is pictured in figure 1. Each group is 104 bits long, divided into 4 blocks of 26 bits length each. The blocks then consist of a 16 bit information word containing the encoded data, and a 10 bit checkword, which is created by calculating a CRC of the information word in addition with a specified offset word (more on that later 3.3). The contents of the actual information parts differ for each group.

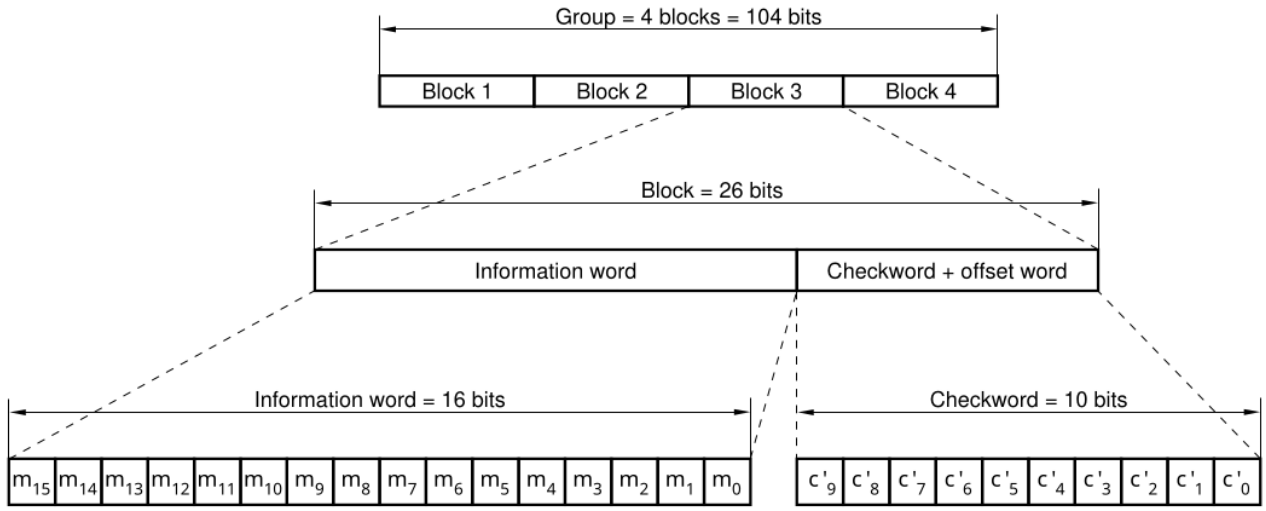

Figure 1: Basic group structure [1]

The aim of this assignment was to create an encoder and a decoder for the Group 0A and 2A message types. The encoder takes input data (such as Program identification, list of Alternative frequencies, etc.) and encodes them into a binary string conforming to the structure of the specified message group. The decoder is then able to take the recieved binary string and decode the original data from it.

1.1 Overview of Group 0A

Group 0A is used for encoding basic tuning and switching information. Specifically it contains the following information [2]:

- Programme identification (PI) – Unique 16-bit code defining the station (every station in the country should have a unique code).
- Group type – Unique 5-bit code defining each group type (00000 for group 0A).
- Programme type (PTY) – 31 pre-defined programme types (genres). For example Rock music, News, etc.
- Traffic programme (TP) and Traffic announcement (TA) – Flags used to signal that the incoming message is a traffic bulletin. So a reciever can, for example, pause a CD so a driver can listen to the news regarding traffic.
- Music/Speech (MS) – Whether there is currently speech or music playing on the station.
- Decoder control bits – Address of the group (0 to 3).
- Alternative Frequencies (AF) – A list of frequencies providing the same station the reciever can re-tune to if the original signal becomes too weak.
- Programme service name (PS) – 2 characters of an 8-character station identity name.

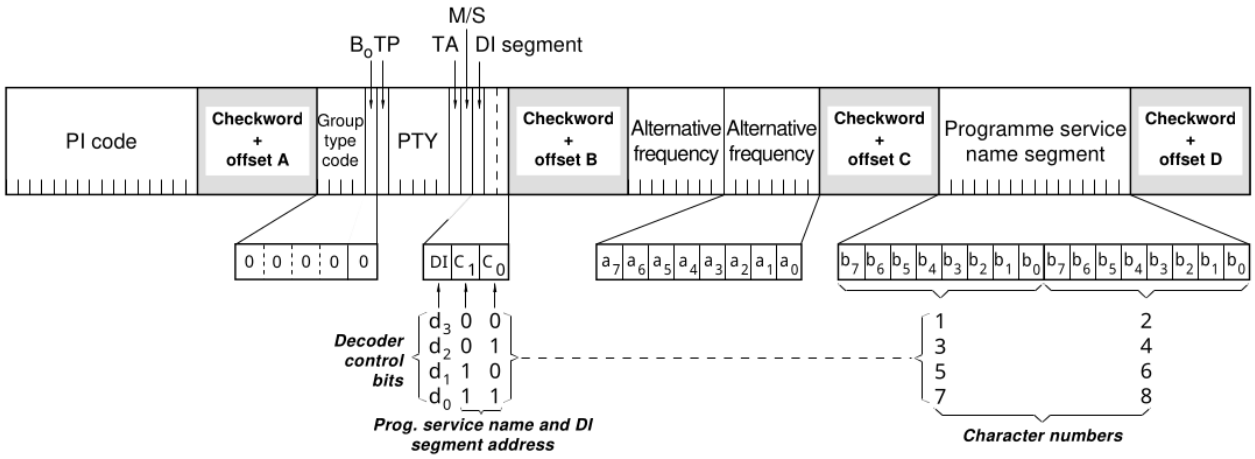

Figure 2: Structure of Group 0A [1]

1.2 Overview of Group 2A

Group 2A is used for encoding radio text, i.e. the message that is displayed on the reciever's display (eg. in someone's car). The group encodes the following information:

- Programme identification (PI) – Unique 16-bit code defining the station (every station in the country should have a unique code).
- Group type – Unique 5-bit code defining each group type (00100 for group 2A).
- Programme type (PTY) – 31 pre-defined programme types (genres). For example Rock music, News, etc.

- Text Segment address code – Address of the group (0 to 15).
- RadioText Segment 1 – 2 characters of a 64-character message.
- RadioText Segment 2 – Another 2 characters of the 64-character message.

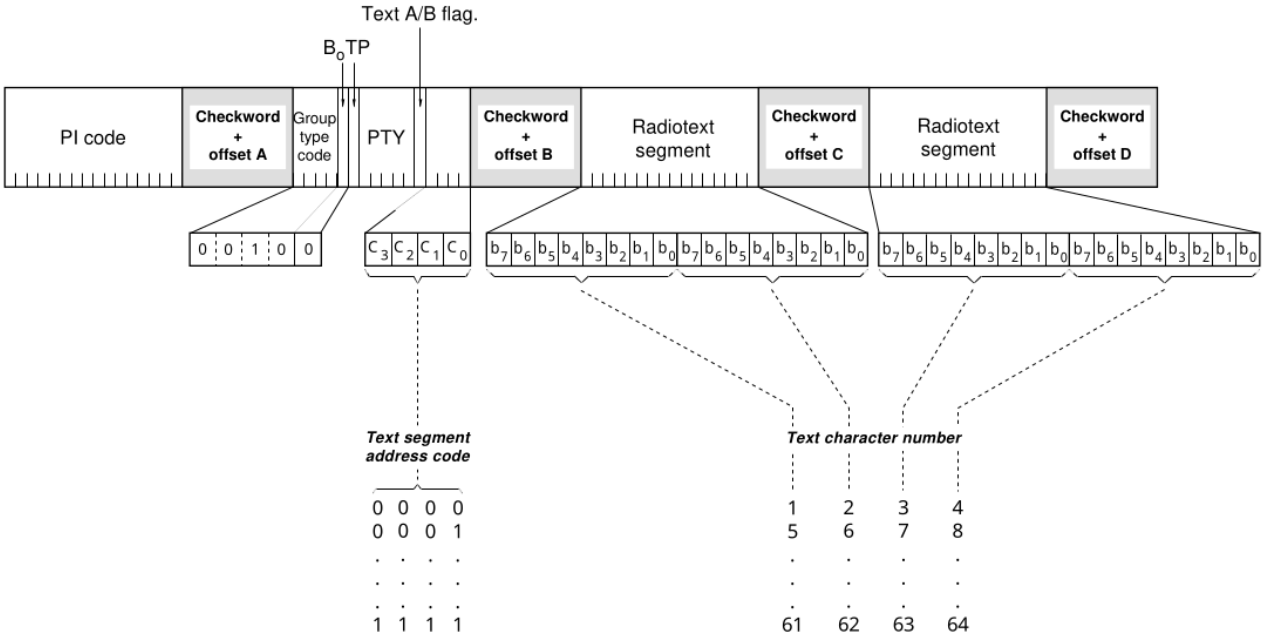

Figure 3: Structure of Group 2A [1]

2 Program usage

This section covers the usage of both the `rds_encoder` and `rds_decoder` programs.

2.1 Compilation

The project can be compiled using the command `make`. The source files are written in C++14, the compiler in use is GCC. The command `make clean` may be used to delete all object and executable files.

2.2 Encoder usage

Encoding of group 0A:

```
./rds_encoder [-g 0A] [-pi <pi>] [-pty <pty>] [-tp <tp>] [-ms <ms>]
[-ta <ta>] [-af <af>] [-ps <ps>] [--help]
```

Encoding of group 2A:

```
./rds_encoder [-g 2A] [-pi <pi>] [-pty <pty>] [-rt <rt>] [-ab <ab>]
[--help]
```

All arguments are **mandatory** (except for `--help`).

Meaning and expected values for each argument:

- -g: Group type (either ”0A” or ”2A”),
- -pi: Programme id (16-bit unsigned int),
- -pty: Programme type (5-bit unsigned int),
- -tp: Traffic programme (boolean, 0 or 1),
- -ms: Music/Speech (boolean, 0 or 1),
- -ta: Traffic announcement (boolean, 0 or 1),
- -af: Alternative frequencies (float,float), values must be between 87.6 and 113.0,
- -ps: Programme service name (max. 8-char long string),
- -rt: Radio text (max. 64-char long string),
- -ab: Radio text A\B (boolean, 0 or 1).

2.3 Decoder usage

```
./rds_encoder [-b <b>] [--help]
```

The -b argument value should contain a binary string that is n*104 bits long and it is also a **mandatory** argument.

3 Implementation

This chapter focuses on describing the implementation of both the encoder and the decoder as well as pointing out some challenging parts of the implementation.

3.1 Implementation of the encoder

First of all comes the parsing of command line arguments. All of the arguments in `argv` are being looped through, parsed, and stored in the `Arguments` object one by one. In the end there are some logical checks to ensure that the input combination of arguments is valid.

After that we may approach the actual encoding part. There are two main classes – `Group0A` and `Group2A`, of which the appropriate one is selected based on the supplied arguments. There is a for loop that loops 4 or 16 times (depending on the group type), in each iteration one group gets encoded (A group 0A message takes 4 groups, while a group 2A message takes 16 groups). The output of each individual group is concatenated to the `output` string and in the end printed on standard output.

Let's take a look at the encoding of one individual 0A group (the encoding of group 2A is analogical, just slightly different data):

The group object (instance of the aforementioned `Group0A` class) has one main method that does all the necessary work – `encode_part(int addr)` and also has the `Arguments` object as a variable. The method goes through the arguments one by one and encodes them into binary strings. For each block it also calculates and encodes the checkwords. The result is then stored in the variable `encoded_string`, which is used in the end when concatenating the results from all the groups.

Overall there were not many challenging parts when implementing the encoder, only the creation of checkwords, which will be described later.

3.2 Implementation of the decoder

The decoding is trickier. Not the parsing of command line arguments tho, the decoder only takes one argument – the input binary string (so we only need to check whether all of its characters are 1's and 0's and that it has a valid length).

After parsing the arguments the next step is to identify which group we should be decoding. For that the first 104 bits (the first group) are used. Again the decoding process is very similar for both groups, so I will only explain the basic process on the example of group 0A:

The first challenging part is looking out for mixed up groups and properly reordering them. For that a couple of arrays are used – boolean `group_recieved[4]` and string `message[4]`. While decoding each group, the address of the group is extracted and used as an index in these two arrays. The `group_recieved` array is initialized to all false, and when a group with say address 1 is decoded, then the flag at index 1 in the array is set to true. That way we can check for duplicate groups as well as whether all groups have been recieved.

The `message` array is used simply to store parts of the message at their actual correct places and then in the end concatenating the parts into the final message.

Another challenging part is deciding whether there are parts of the message missing. That cannot be done by simply checking if the decoded string is 4*104 chars long, because shorter messages are also valid. Instead a simple algorithm is used that once again utilizes the `group_recieved` boolean

array. We can loop through the array and look for the first encountered false flag. Once encountered, there may be no true flags in the rest of the array, otherwise it will be clear that a part of the originally sent message has been lost (eg. [true, false, true, true] indicates that the second part was not recieved).

When decoding one specific group, another challenging part is to detect any mixed up blocks and correctly reorder them. For that we can use the checkwords of each block. The program splits the group into four blocks and iterates through them one by one. In each block it calculates the checkword for the information part of the block (the exact same process that happens during encoding – that way we can look for any differences) and compares it with the actual recieved checkword. If they don't match, it tries again with a different offset word and if none of the offset words create a proper checkword, it is clear that some part of the message has been corrupted.

Anyways, depending on which offset word works to create the actual checkword, we can determine which part of the message we are actually looking at (for example looking at the first 26 bits but the checkword corresponds to offset word 'C' - so we are actually looking at the third block and can reorder it in a similar way to the groups reordering).

After reordering all the groups and blocks the process remains relatively simple. The first group that we decoded way back at the start serves as a sort of a standard for the remaining groups. All the decoded data (apart from the group address and the text part of course) are compared to the data of the first group and if there are any differences, then there has clearly been some sort of an error during transmission of the message and we can print an error.

In the end all there's left to do is to concatenate all the decoded text parts and print the decoded info.

3.3 Checkword creation

Cyclic redundancy check (CRC) is used in RDS, much like in many other networking standards and protocols, to ensure data integrity. It works by creating a unique checksum for the information part of each block. Thanks to that we can then detect and even to some extent repair errors which occur during transmission. However in this project the CRC is only used to detect errors, repairing is not implemented.

As mentioned previously, every group in RDS is divided into 4 blocks, 26 bits long each. Each block then contains 16 bits of information and a 10-bit checkword. The checkword is created by calculating the CRC of the information part and adding the result to a predetermined offset word. Each block uses a different offset word. Thanks to that we can detect any block mix ups within groups.

The CRC algorithm operates using polynomial division in binary form. The process begins with appending a predetermined number of zero bits to the information part of the block. This number corresponds to the degree of the generator polynomial used for the CRC. The polynomial used in RDS is $x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$. The binary data concatenated with zero bits is then divided by the generator polynomial using modulo-2 division, where no carries are involved. The division results in a remainder, which serves as the CRC value.

All the bitwise operations are implemented on strings of 1's and 0's rather than on bitsets or binary vectors, because it's not really any more difficult using just strings (XOR implementation is trivial). Parts of the implementation are presented in the screenshots below (fig. 4, 5, 6).

```
string modulo2_division(string dividend, string divisor) {
    string remainder = dividend;

    //iterate over the dividend
    for (u_int64_t i = 0; i <= remainder.length() - divisor.length(); ++i) {
        //if the current bit is 1, perform the XOR operation
        if (remainder[i] == '1') {
            for (u_int64_t j = 0; j < divisor.length(); ++j) {
                remainder[i + j] = remainder[i + j] == divisor[j] ? '0' : '1';
            }
        }
    }

    return remainder.substr(remainder.length() - divisor.length() + 1);
}
```

Figure 4: Modulo 2 division

```
string calculate_crc(string input) {
    //x^10 + x^8 + x^7 + x^5 + x^4 + x^3 + 1
    string generator = "00110111001";
    int gen_length = generator.length();

    //pad the input with zeros
    string padded_input = input + string(gen_length - 1, '0');

    //calculate the remainder
    string remainder = modulo2_division(padded_input, generator);

    return remainder;
}
```

Figure 5: CRC calculation

```
string create_checkword(string crc, string offset_type) {
    string checkword = "";
    map<string, string> offset_words = {
        {"A", "0011111100"},
        {"B", "0110011000"},
        {"C", "0101101000"},
        {"D", "0110110100"}
    };

    string block_offset = offset_words[offset_type];

    //xor the CRC with the offset word
    for (u_int64_t i = 0; i < crc.length(); i++) {
        checkword += crc[i] == block_offset[i] ? '0' : '1';
    }

    return checkword;
}
```

Figure 6: Final checkword creation

4 Testing

This section presents various test cases for both the encoder and the decoder. This is only a selection of the most important ones, the complete lists of all used test cases can be found in the `*_test_inputs.txt` files in the root directory of the project.

For example the decoder tests presented here are only for Group 0A, but the same exact tests were performed for Group 2A and all passed.

4.1 Valid encoder inputs

Correct Group 0A input:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 104.5,98.0 -ps "RadioXYZ"
0001001000110100000011010100000001001011000011111111101010101001101001000000110110101001001100001101010100100010010001101
0000011010100000001001011000110010001110000000000000000010110100001100100011010011111000110000100100011010000001101010000
0001001011001000100011000000000000000000001011010000110111101011000010011101000010010001101000000110101000000010010110011
0100110101000000000000000000010110100001011001010110100000100100
```

Correct Group 0A input with shorter message (adds padding spaces):

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 104.5,98.0 -ps "Radio"
0001001000110100000011010100000001001011000011111111101010101001101001000000110110101001001100001101010100100010010001101
0000011010100000001001011000110010001110000000000000000010110100001100100011010011111000110000100100011010000001101010000
0001001011001000100011000000000000000000001011010000110111100100000110111011000010010001101000000110101000000010010110011
0100110101000000000000000000010110100000100000001000000011011100
```

Correct Group 2A input:

```
[michalb@fedora proj]$ ./rds_encoder -g 2A -pi 4660 -pty 5 -tp 1 -rt "Now Playing Song Title by Artist" -ab 0
0001001000110100000011010100010010010100000111110111001001110011011111001111011011101110010000011001001000000100100011101
0000011010100010010010100001100101011101010000011011001000101010011000010111100111101010100010010001101000000110101000
1001001010001000100111000110100101101111010011110010110011100100000000010111100010010001101000000110101000100100101000111
0100100101010100110110111110110000101011011100110011110000010110001001000110100000110101000100100101001000010110011001
0000010101001000010010011010010111010001011101011000100100011010000011010100010010010100101010000101001101110001100101
0001110100100000011000101110011001000100100011010000011010100010010010100110111100000101111001001000000100110100101000
0101110010011011001000010010001101000001101010001001001010011111001111100001110100011010010000001000101110011011101000010
0000010001001000110100000110101000100100101010000011101101001000000000000000000000000000100000000100000000110111000000100
0011010000011010100010010010101001010101010000100000001000000000000000000000100000000100000000110111000000100100011010000110
1010001001001010101011100111111001000000010000000000000000000010000000010000000110111000000100100011010000011010100010010010
1010111000100110001000000001000000000000000000010000000010000000110111000000100100011010000011010100010010010101011001110110
0000100000000100000000000000000010000000010000000110111000000100100011010000011010100010010010101101100000100100100000010
0000000000000010000000100000001101110000001001000110100000110101000100100101011100011000010001000000001000000000000000
00100000000100000001101110000001001000110100000110101000100100101011110101111011001000000010000000000000000000100000001000
000011011100
```

4.2 Invalid encoder inputs

Missing arguments:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pty 5 -tp 1 -ms 0 -af 104.5,98.0 -ps "RadioXYZ"
Error: Invalid arguments. Use --help for more information
```

Message too long:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 104.5,98.0 -ps "RadioABCD"
Error: Message too long
```

PI value out of range:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 65536 -pty 5 -tp 1 -ms 0 -ta 1 -af 104.5,98.0 -ps "RadioXYZ"
Error: Argument value out of range
```

Invalid AF format:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 98.5-104.5 -ps "RadioXYZ"
Error: Invalid alternative frequencies format
```

AF out of range:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 10.0,98.0 -ps "RadioXYZ"
Error: The lowest allowed alternative frequency is 87.6 MHz
```

Invalid group:

```
[michalb@fedora proj]$ ./rds_encoder -g 1A -pi 4660 -pty 5 -tp 1 -ms 0 -ta 1 -af 104.5,98.0 -ps "RadioXYZ"
Error: Group (-g) must be 0A or 2A
```

4.3 Valid decoder inputs

Groups and blocks in the right order:

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010101010011010010000011011
0101001001100001101010100100010010001101000001101010000001001011000110010001110000000000000000010110100001100100011010
0111110001100001001000110100000110101000000100101100100010001100000000000000000010110100001101111010110000100111010000
0100100011010000011010100000010010110011010011010100000000000000000101101000010110010101101000001001 00
PI: 4660
GT: 0A
TP: 1
PTY: 5
TA: Active
MS: Speech
DI: 0
AF: 104.5, 98.0
PS: "RadioXYZ"
```

Mixed blocks within groups, group order correct:

```
[michalb@fedora proj]$ ./rds_decoder -b 0000010010110000111111111000010010001101000001101010101010100110100100000110110
0101001001100001101010100100010010001101000001101010000001001011000110010001110110010001101001111100011000000000000000
0001011010000001001000110100000110101000000100101100100010001100000000000000000010110100001101111010110000100111010000
0100100011010000011010100000010010110011010011010100000000000000000101101000010110010101101000001001 00
PI: 4660
GT: 0A
TP: 1
PTY: 5
TA: Active
MS: Speech
DI: 0
AF: 104.5, 98.0
PS: "RadioXYZ"
```

Mixed groups (0 2 1 3), block order correct:

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010101010011010010000011011
0101001001100001101010100100010010001101000001101010000001001011001000100011000000000000000000001011010000110111101011
0001001110100001001000110100000110101000000100101100011001000111000000000000000001011010000110010001101001111100011000
0100100011010000011010100000010010110011010011010100000000000000001011010000101100101011010000100100
PI: 4660
GT: 0A
TP: 1
PTY: 5
TA: Active
MS: Speech
DI: 0
AF: 104.5, 98.0
PS: "RadioXYZ"
```

Mixed groups (0 2 1 3), mixed blocks within groups:

```
[michalb@fedora proj]$ ./rds_decoder -b 0000010010110000111111111010101010011010010000011011000100100011010000011010100
0101001001100001101010100100010010001101000001101010000000000000000001011010000000010010110010001000110001101111010110
0001001110100001001000110100000110101000000100101100011001000111000000000000000001011010000110010001101001111100011000
0100100011010000011010100000010010110011010011010100000000000000001011010000101100101011010000100100
PI: 4660
GT: 0A
TP: 1
PTY: 5
TA: Active
MS: Speech
DI: 0
AF: 104.5, 98.0
PS: "RadioXYZ"
```

Recieved message shorter than max. length (rec. groups 0 1 2):

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010101010011010010000011011
0101001001100001101010100100010010001101000001101010000001001011000110010001110000000000000000001011010000110010001101
0111110001100001001000110100000110101000000100101100010001000110000000000000000001011010000110111101011000010011101
PI: 4660
GT: 0A
TP: 1
PTY: 5
TA: Active
MS: Speech
DI: 0
AF: 104.5, 98.0
PS: "RadioX"
```

4.4 Invalid decoder inputs

Invalid input string length:

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010111010101001000100100011
0100000110101000000100101100011001000111000000000000000001011010000110010001101001111100011000010010001101000001101010
0000010010110011000100011000000000000000000001011010000110111101011000010011101000010010001101000001101010000001001011100
1101001101010000000000000000001011010000101100101011010000100100
Error: Input length is not a multiple of 104.
```

Failed CRC calculation:

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010101010011010010000011011
0101001001100001101010100100010010001101000001101010000001001011000110110001110000000000000000001011010000110010001101
0111110001100001001000110100000110101000000100101100010001000110000000000000000001011010000110111101011000010011101000
0100100011010000011010100000010010110011010011010100000000000000001111000100101100101011010000100100
Error: Message is corrupted (crc check failed)
```

Recieved groups 0 1 and 3 (2 missing):

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100001111111111010101010011010010000011011
0101001001100001101010100100010001000110100000110101000000100101100011001000111000000000000000000010110100001100100011010
0111110001100001001000110100000110101000000100101100110100110101000000000000000000010110100001011001010110100000100100
Error: Missing part of message
```

Duplicate group address:

```
[michalb@fedora proj]$ ./rds_decoder -b 0001001000110100000110101000000100101100011001000111101010100110100100000110110
0101001001100001101010100100010001000110100000110101000000100101100011001000111000000000000000000010110100001100100011010
0111110001100001001000110100000110101000000100101100100010001100000000000000000010110100001101111010110000100111010000
0100100011010000011010100000010010110011010011010100000000000000000010110100001011001010110100000100100
Error: Duplicate group detected (two groups with the same address)
```

4.5 Combined tests

Group 0A encoding, then decoding:

```
[michalb@fedora proj]$ ./rds_encoder -g 0A -pi 1234 -pty 10 -tp 1 -ms 1 -ta 0 -af 100.2,97.4 -ps "Test"
0000010011010010110001100000000101010010000100000000011111111011000111001011101010100011001011100111100000001001101000
1011000110000000010101001001001011100100000000000000000010110100001110011011101000010000001000001001101001011000110000
0001010100101010011100100000000000000000010110100000100000001000000011011100000000100110100101100011000000001010100101 1
1111001011000000000000000001011010000010000000100000001101110000000100110100101100011000000001010100101 1
[michalb@fedora proj]$ ./rds_decoder -b 0000010011010010110001100000000101010010000100000000011111111011000111001011101
0101010001100101111001111000000010011010010110001100000000101010010010011011100100000000000000000010110100001110011011101
0000100000010000010011010010110001100000000101010010101001110010000000000000000010110100000100000001000000011011100000
0001001101001011000110000000010101001011111100101100000000000000000010110100000100000001000000011011100
PI: 1234
GT: 0A
TP: 1
PTY: 10
TA: Inactive
MS: Music
DI: 0
AF: 100.2, 97.4
PS: "Test"
```

Group 2A encoding, then decoding:

```
[michalb@fedora proj]$ ./rds_encoder -g 2A -pi 3333 -pty 3 -tp 1 -rt "Now playing The Emptiness Machine by Linkin Park
" -ab 1
00001101000000101101110110100100100011100000000000100001001110011011111100111101101110111001000001100100100000011010000001
01101110110100100100011100010110110001011100000011011000110000101011000010111100111110101010000110100000101101110110100
10010001110010110111101001101001011011101001111001011001110010000000001011110000110100000101101110110100100100011100 11
10110000110101010001101000100000011101100101001000000110010111000011010000001011011101101001001000111010011010101010100
01010110110111001101000111000000111010001111001010000110100000010110111011010010010001110101101110110001101001011011 1010
01111001011001010111001100110101000000110100000010110111011010010010001110110000001001110111001100100000000100010000 10011
01011000010011101111000011010000001011011101101001001000111011101100111100110001101101000000101101110110100101101110 1010
100101000011010000001011011101101001001000111100011000010110110010100100000001010010110110001001111001101011000100001101
000000101101110110100100100011110011010110010001000000010011001010101110011010010110111010101001010000110100000010110 1110
1101001001000111101000011110010110101101101001100101011101101110001000000111010101000001101000000101101110110100 10010001
11101101110000000101000000110000111110011010111001001101011010111011100001101000000101101110110100100100011111000000 10101
1000100000000100000000000000000001000000001000000001101110000000110100000010110111011010010010001111101011110111100100000 0010
00000000000000000100000000100000000110111000000011010000001011011101101001001000111111011001001000010000000010000000000000 000
0010000000001000000001101110000000110100000010110111011010010010001111111101001110100100000000100000000000000000100000000 1000
000011011100
[michalb@fedora proj]$ ./rds_decoder -b 0000110100000010110111011010010010001110000000000010000100111001101111110011101 1
01110111001000001100100100000011010000001011011101101001001000111000101101100010111000000110110001100001010110000101111 10
0111110101010000110100000010110111011010010010001110010110111101001101001011011101001111001011001110010000000001011 1100
00110100000010110111011010010010001110011101100001101010100011010001000000111011001010010000001100101110000110100000 0101
10111011010010010001110100110101010101000101011011011100110100011100000011101000111100101000011010000001011011101101 0010
010001110101101110110001101001011011101001111001011001010111001100110101000000110100000010110111011010010010001110 11000
0010011101110011001000000001000100001001101011000010011101111000011010000001011011101101001001000111011101100111100 11000
11011010000001011011101101001011011101010100101000011010000001011011101101001001000111100011000010110110010100100000 00101
00101101100010011110011010110001000011010000001011011101101001001000111100110101100100010000000100110010101011100110 1001
0110111010101001010000110100000010110111011010010010001111010000111100101101011011010011001010111011011100010000001 11010
10100000011010000001011011101101001001000111101101110000000101000000110000111110011010111001001101011010111011100001 10100
00010110111011010010010001111100000010101100010000000010000000000000000010000000010000000011011100000001101000000101101 11011
0100100100011111010111101111001000000001000000000000000001000000001000000001101110000000110100000010110111011010010010001 11
11101100100100001000000001000000000000000001000000001000000001101110000000110100000010110111011010010010001111111101001 1101
00100000000100000000000000000100000000100000000 11011100
PI: 3333
GT: 2A
TP: 1
PTY: 3
A/B: 1
RT: "Now playing The Emptiness Machine by Linkin Park"
```

5 References

[1] CENELEC. *EN50067 RDS Standard*. 1998. URL: http://www.interactive-radio-system.com/docs/EN50067_RDS_Standard.pdf. visited on 10/12/2024.

[2] *Wikipedia: Radio Data System*. URL: https://en.wikipedia.org/wiki/Radio_Data_System. visited on 10/12/2024.