Reed-Solomon Library Programming Interface

Thomas Gleixner

tglx@linutronix.de

Reed-Solomon Library Programming Interface

by Thomas Gleixner

Table of Contents

Chapter 1. Introduction

The generic Reed-Solomon Library provides encoding, decoding and error correction functions.

Reed-Solomon codes are used in communication and storage applications to ensure data integrity.

This documentation is provided for developers who want to utilize the functions provided by the library.

Chapter 2. Known Bugs And Assumptions

None.

Chapter 3. Usage

This chapter provides examples of how to use the library.

3.1. Initializing

The init function init_rs returns a pointer to an rs decoder structure, which holds the necessary information for encoding, decoding and error correction with the given polynomial. It either uses an existing matching decoder or creates a new one. On creation all the lookup tables for fast en/decoding are created. The function may take a while, so make sure not to call it in critical code paths.

```
/* the Reed Solomon control structure */
static struct rs_control *rs_decoder;

/* Symbolsize is 10 (bits)
 * Primitive polynomial is x^10+x^3+1
 * first consecutive root is 0
 * primitive element to generate roots = 1
 * generator polynomial degree (number of roots) = 6
 */
rs_decoder = init_rs (10, 0x409, 0, 1, 6);
```

3.2. Encoding

The encoder calculates the Reed-Solomon code over the given data length and stores the result in the parity buffer. Note that the parity buffer must be initialized before calling the encoder.

The expanded data can be inverted on the fly by providing a non-zero inversion mask. The expanded data is XOR'ed with the mask. This is used e.g. for FLASH ECC, where the all 0xFF is inverted to an all 0x00. The Reed-Solomon code for all 0x00 is all 0x00. The code is inverted before storing to FLASH so it is 0xFF too. This prevents that reading from an erased FLASH results in ECC errors.

The databytes are expanded to the given symbol size on the fly. There is no support for encoding continuous bitstreams with a symbol size != 8 at the moment. If it is necessary it should be not a big deal to implement such functionality.

```
/* Parity buffer. Size = number of roots */
uint16_t par[6];
/* Initialize the parity buffer */
memset(par, 0, sizeof(par));
/* Encode 512 byte in data8. Store parity in buffer par */
encode_rs8 (rs_decoder, data8, 512, par, 0);
```

3.3. Decoding

The decoder calculates the syndrome over the given data length and the received parity symbols and corrects errors in the data.

If a syndrome is available from a hardware decoder then the syndrome calculation is skipped.

The correction of the data buffer can be suppressed by providing a correction pattern buffer and an error location buffer to the decoder. The decoder stores the calculated error location and the correction bitmask in the given buffers. This is useful for hardware decoders which use a weird bit ordering scheme.

The databytes are expanded to the given symbol size on the fly. There is no support for decoding continuous bitstreams with a symbolsize != 8 at the moment. If it is necessary it should be not a big deal to implement such functionality.

3.3.1. Decoding with syndrome calculation, direct data correction

```
/* Parity buffer. Size = number of roots */
uint16_t par[6];
uint8_t  data[512];
int numerr;
/* Receive data */
.....
/* Receive parity */
.....
/* Decode 512 byte in data8.*/
numerr = decode_rs8 (rs_decoder, data8, par, 512, NULL, 0, NULL, 0, NULL);
```

3.3.2. Decoding with syndrome given by hardware decoder, direct data correction

```
/* Parity buffer. Size = number of roots */
uint16_t par[6], syn[6];
uint8_t  data[512];
int numerr;
/* Receive data */
.....
/* Receive parity */
```

```
.....
/* Get syndrome from hardware decoder */
.....
/* Decode 512 byte in data8.*/
numerr = decode_rs8 (rs_decoder, data8, par, 512, syn, 0, NULL, 0, NULL);
```

3.3.3. Decoding with syndrome given by hardware decoder, no direct data correction.

Note: It's not necessary to give data and received parity to the decoder.

```
/* Parity buffer. Size = number of roots */
uint16_t par[6], syn[6], corr[8];
uint8_t  data[512];
int numerr, errpos[8];
/* Receive data */
.....
/* Receive parity */
.....
/* Get syndrome from hardware decoder */
.....
/* Decode 512 byte in data8.*/
numerr = decode_rs8 (rs_decoder, NULL, NULL, 512, syn, 0, errpos, 0, corr);
for (i = 0; i < numerr; i++) {
 do_error_correction_in_your_buffer(errpos[i], corr[i]);
}
```

3.4. Cleanup

The function free_rs frees the allocated resources, if the caller is the last user of the decoder.

```
/* Release resources */
free_rs(rs_decoder);
```

Chapter 4. Structures

This chapter contains the autogenerated documentation of the structures which are used in the Reed-Solomon Library and are relevant for a developer.

struct rs_control

LINUX

Kernel Hackers ManualApril 2009

Name

`struct rs_control` — rs control structure

Synopsis

```
struct rs_control {
  int mm;
  int nn;
  uint16_t * alpha_to;
  uint16_t * index_of;
  uint16_t * genpoly;
  int nroots;
  int fcr;
  int prim;
  int iprim;
  int gfpoly;
  int (* gffunc) (int);
  int users;
  struct list_head list;
};
```

Members

mm

Bits per symbol

nn

Symbols per block (= (1<<mm)-1)

alpha_to

log lookup table

index_of

Antilog lookup table

genpoly

Generator polynomial

nroots

Number of generator roots = number of parity symbols

fcr

First consecutive root, index form

prim

Primitive element, index form

iprim

prim-th root of 1, index form

gfpoly

The primitive generator polynominal

gffunc

Function to generate the field, if non-canonical representation

users

Users of this structure

list

List entry for the rs control list

Chapter 5. Public Functions Provided

This chapter contains the autogenerated documentation of the Reed-Solomon functions which are exported.

free_rs

LINUX

Kernel Hackers ManualApril 2009

Name

`free_rs` — Free the rs control structure, if it is no longer used

Synopsis

```
void free_rs (struct rs_control * rs);
```

Arguments

rs

the control structure which is not longer used by the caller

init_rs

LINUX

Kernel Hackers ManualApril 2009

Name

`init_rs` — Find a matching or allocate a new rs control structure

Synopsis

```
struct rs_control * init_rs (int symsize, int gfpoly, int fcr, int prim, int
nroots);
```

Arguments

symsize

the symbol size (number of bits)

gfpoly

the extended Galois field generator polynomial coefficients, with the 0th coefficient in the low order bit. The polynomial must be primitive;

fcr

the first consecutive root of the rs code generator polynomial in index form

prim

primitive element to generate polynomial roots

nroots

RS code generator polynomial degree (number of roots)

init_rs_non_canonical

LINUX

Kernel Hackers ManualApril 2009

Name

`init_rs_non_canonical` — Find a matching or allocate a new rs control

Synopsis

```
struct rs_control * init_rs_non_canonical (int symsize, int (*gffunc) (int),
int fcr, int prim, int nroots);
```

Arguments

symsize

the symbol size (number of bits)

gffunc

pointer to function to generate the next field element, or the multiplicative identity element if given 0. Used instead of gfpoly if gfpoly is 0

fcr

the first consecutive root of the rs code generator polynomial in index form

prim

primitive element to generate polynomial roots

nroots

RS code generator polynomial degree (number of roots)

Description

structure, for fields with non-canonical representation

encode_rs8

LINUX

Kernel Hackers ManualApril 2009

Name

`encode_rs8` — Calculate the parity for data values (8bit data width)

Synopsis

```
int encode_rs8 (struct rs_control * rs, uint8_t * data, int len, uint16_t *
par, uint16_t invmsk);
```

Arguments

rs

the rs control structure

data

data field of a given type

len

data length

par

parity data, must be initialized by caller (usually all 0)

invmsk

invert data mask (will be xored on data)

Description

The parity uses a uint16_t data type to enable symbol size > 8. The calling code must take care of encoding of the syndrome result for storage itself.

decode_rs8

LINUX

Kernel Hackers ManualApril 2009

Name

`decode_rs8` — Decode codeword (8bit data width)

Synopsis

```
int decode_rs8 (struct rs_control * rs, uint8_t * data, uint16_t * par, int
len, uint16_t * s, int no_eras, int * eras_pos, uint16_t invmsk, uint16_t *
corr);
```

Arguments

rs

the rs control structure

data

data field of a given type

par

received parity data field

len

data length

s

syndrome data field (if NULL, syndrome is calculated)

no_eras

number of erasures

eras_pos

position of erasures, can be NULL

invmsk

invert data mask (will be xored on data, not on parity!)

corr

buffer to store correction bitmask on eras_pos

Description

The syndrome and parity uses a uint16_t data type to enable symbol size > 8. The calling code must take care of decoding of the syndrome result and the received parity before calling this code. Returns the number of corrected bits or -EBADMSG for uncorrectable errors.

encode_rs16

LINUX

Name

`encode_rs16` — Calculate the parity for data values (16bit data width)

Synopsis

```
int encode_rs16 (struct rs_control * rs, uint16_t * data, int len, uint16_t *
par, uint16_t invmsk);
```

Arguments

rs

the rs control structure

data

data field of a given type

len

data length

par

parity data, must be initialized by caller (usually all 0)

invmsk

invert data mask (will be xored on data, not on parity!)

Description

Each field in the data array contains up to symbol size bits of valid data.

decode_rs16

LINUX

Name

decode_rs16 — Decode codeword (16bit data width)

Synopsis

```
int decode_rs16 (struct rs_control * rs, uint16_t * data, uint16_t * par, int
len, uint16_t * s, int no_eras, int * eras_pos, uint16_t invmsk, uint16_t *
corr);
```

Arguments

rs

the rs control structure

data

data field of a given type

par

received parity data field

len

data length

s

syndrome data field (if NULL, syndrome is calculated)

no_eras

number of erasures

eras_pos

position of erasures, can be NULL

invmsk

invert data mask (will be xored on data, not on parity!)

corr

buffer to store correction bitmask on eras_pos

Description

Each field in the data array contains up to symbol size bits of valid data. Returns the number of corrected bits or -EBADMSG for uncorrectable errors.

Chapter 6. Credits

The library code for encoding and decoding was written by Phil Karn.

```
Copyright 2002, Phil Karn, KA9Q
 May be used under the terms of the GNU General Public License (GPL)
```

The wrapper functions and interfaces are written by Thomas Gleixner.

Many users have provided bugfixes, improvements and helping hands for testing. Thanks a lot.

The following people have contributed to this document:

Thomas Gleixner<`tglx@linutronix.de`>