Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Open Source Version

API Reference Manual - Version 2.19.0

July 27, 2017

Contents

CHAPTER 1
STORAGE LIBRARY

1.1 About This Document

This document describes the software programming interface and operation of functions in the library. Sections in this document are grouped by the functions found in individual header files that define the function prototypes. Subsections include function parameters, description and type.

This document refers to the open release version of the library. A separate, crypto release called the Intel® Intelligent Storage Acceleration Library Crypto (Intel® ISA-L Crypto) is also available and contains an extended set of functions.

1.2 Overview

The Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Open Source Version is a collection of functions used in storage applications optimized for Intel architecture Intel® 64. In some cases, multiple versions of the same function are available that are optimized for a particular Intel architecture and instruction set. This software takes advantage of new instructions and users should ensure that the chosen function is compatible with hardware it will run on.

Multibinary support has been added for many units in ISA-L. With multibinary support functions, an appropriate version is selected at first run and can be called instead of the architecture-specific versions. This allows users to deploy a single binary with multiple function versions and choose at run time based on platform features. Users can still call the architecture-specific versions directly to reduce code size. There are also base functions, written in C, which the multibinary function will call if none of the required instruction sets are enabled.

1.3 RAID Functions

Functions in the RAID section calculate and operate on XOR and P+Q parity found in common RAID implementations. The mathematics of RAID are based on Galois finite-field arithmetic to find one or two parity bytes for each byte in N sources such that single or dual disk failures (one or two erasures) can be corrected. For RAID5, a block of parity is calculated by the xor across the N source arrays. Each parity byte is calculated from N sources by:

$$P = D_0 + D_1 + ... + D_{N-1}$$

where D_n are elements across each source array [0-(N-1)] and + is the bit-wise exclusive or (xor) operation. Elements in GF(2^8) are implemented as bytes.

For RAID6, two parity bytes P and Q are calculated from the source array. P is calculated as in RAID5 and Q is calculated using the generator g as:

$$Q = g^0 D_0 + g^1 D_1 + g^2 D_2 + ... + g^{N-1} D_{N-1}$$

where g is chosen as {2}, the second field element. Multiplication and the field are defined using the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (0x1d).

1.4 Erasure Code Functions

Functions pertaining to erasure codes implement a general Reed-Solomon type encoding for blocks of data to protect against erasure of whole blocks. Individual operations can be described in terms of arithmetic in the Galois finite field GF(2^8) with the particular field-defining primitive or reducing polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (0x1d).

For example, the function ec_encode_data() will generate a set of parity blocks P_i from the set of k source blocks D_i and arbitrary encoding coefficients $a_{i,j}$ where each byte in P is calculated from sources as:

$$P_i = \sum_{j=1}^{k} a_{i,j} \cdot D_j$$

where addition and multiplication $\cdot$ is defined in GF(2^8). Since any arbitrary set of coefficients $a_{i,j}$ can be supplied, the same fundamental function can be used for encoding blocks or decoding from blocks in erasure.

1.5 CRC Functions

Functions in the CRC section are fast implementations of cyclic redundancy check using IA specialized instructions such as PCLMULQDQ, carry-less multiplication. Generally, a CRC is the remainder in binary division of a message and a CRC polynomial in GF(2).

$$CRC(M(x)) = x^{deg(P(x))} \cdot M(x) \, mod \, P(x)$$

CRC is used in many storage applications to ensure integrity of data by appending the CRC to a message. Various standards choose the polynomial P and may vary by initial seeding value, bit reversal and inverting the CRC for example.

1.6 Alignment for Input Parameters

The alignment required for the input parameters of each of the Intel® ISA-L functions is documented in the relevant sections of this API manual. The table below outlines these requirements.

Function	Alignment Required
CRC	No
Erasure Code	32B for gf_vect_mul, none otherwise
RAID	32B or 16B
Igzip	No

1.7 System Requirements

Individual functions may have various run-time requirements such as the minimum version of SSE as described in Instruction Set Requirements. General requirements are listed below.

Recommended Hardware:

- em64t: A system based on the Intel® Xeon® processor with Intel® 64 architecture.
- IA32: When available for 32-bit functions; A system based on the Intel® Xeon® processor or subsequent IA-32 architecture based processor.

Software Requirements:

Most functions in the library use the 64-bit embedded and Unix standard for calling convention http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.95.pdf. When available, 32-bit versions use cdecl. Individual functions are written to be statically linked with an application.

Building Library Functions:

- Yasm Assembler: version at least v1.2.0. (v1.3.0 would be better)
- or Nasm Assembler: version at least v2.10.

Building Examples and Tests:

Examples and test source follow simple command line POSIX standards and should be portable to any mostly POSIX-compliant OS.

Note

Please note that the library assumes 1MB = 1,000,000 bytes in reported performance figures.

CHAPTER 2
FUNCTION VERSION NUMBERS

2.1 Function Version Numbers

Individual functions are given version numbers with the format mm-vv-ssss.
- mm = Two hex digits indicating the processor a function was optimized for.

  - 00 = Nehalem/Jasper Forest/Multibinary
  - 01 = Westmere
  - 02 = Sandybridge
  - 03 = Ivy Bridge
  - 04 = Haswell
  - 05 = Silvermont
  - 06 = Skylake
  - 07 = Goldmont
  - 08 = Cannonlake

- vv = function version number
- ssss = function serial number

2.2 Function Version Numbers Tables

Function	Version
crc16_t10dif_01	01-06-0010
crc32_ieee_01	01-06-0011
crc32_iscsi_simple	00-02-0012
crc32_iscsi_baseline	00-02-0013
crc32_iscsi_00	00-03-0014
crc32_iscsi_01	01-03-0015
crc16_t10dif_by4	05-02-0016
crc32_ieee_by4	05-02-0017
crc64_ecma_norm	00-00-0018
crc64_ecma_norm_base	00-00-0019
crc64_ecma_norm_by8	01-00-001a
crc64_ecma_refl	00-00-001b
crc64_ecma_refl_base	00-00-001c
crc64_ecma_refl_by8	01-00-001d
crc64_iso_norm	00-00-001e
crc64_iso_norm_base	00-00-001f
crc64_iso_norm_by8	01-00-0020
crc64_iso_refl	00-00-0021
crc64_iso_refl_base	00-00-0022
crc64_iso_refl_by8	01-00-0023
crc64_jones_norm	00-00-0024
crc64_jones_norm_base	00-00-0025
crc64_jones_norm_by8	01-00-0026
crc64_jones_refl	00-00-0027
crc64_jones_refl_base	00-00-0028
crc64_jones_refl_by8	01-00-0029
crc32_gzip_refl	00-00-002a
crc32_gzip_refl_base	00-00-002b
crc32_gzip_refl_by8	01-00-002c
xor_gen_sse	00-0c-0030
xor_check_sse	00-03-0031
pq_gen_sse	00-09-0032
pq_check_sse	00-06-0033
gf_vect_mul_sse	00-03-0034
gf_vect_mul_init	00-02-0035
gf_vect_mul_avx	01-03-0036
xor_gen_avx	02-05-0037
pq_gen_avx	02-0a-0039
pq_gen_avx2	04-03-0041
gf_vect_dot_prod_sse	00-05-0060
gf_vect_dot_prod_avx	02-05-0061

Function	Version
gf_2vect_dot_prod_sse	00-04-0062
gf_3vect_dot_prod_sse	00-06-0063
gf_4vect_dot_prod_sse	00-06-0064
gf_5vect_dot_prod_sse	00-05-0065
gf_6vect_dot_prod_sse	00-05-0066
ec_init_tables	00-01-0068
ec_encode_data_sse	00-02-0069
isal_deflate_init	01-03-0081
isal_deflate	01-03-0082
isal_deflate_stateless	01-01-0083
isal_deflate_stateless_init	00-01-0084
isal_update_histogram	00-01-0085
isal_create_hufftables	00-01-0086
isal_create_hufftables_subset	00-01-0087
isal_inflate_init	00-01-0088
isal_inflate_stateless	00-01-0089
isal_inflate	00-01-008a
isal_deflate_set_hufftables	00-01-008b
isal_deflate_set_dict	00-01-008c
isal_inflate_set_dict	00-01-008d
isal_deflate_reset	00-01-008e
isal_inflate_reset	00-01-008f
crc16_t10dif	00-03-011a
crc32_ieee	00-03-011b
crc32_iscsi	00-03-011c
crc32_iscsi_base	00-01-011d
crc16_t10dif_base	00-01-011e
crc32_ieee_base	00-01-011f
xor_gen	00-03-0126
xor_check	00-03-0127
pq_gen	00-03-0128
pq_check	00-03-0129
pq_gen_base	00-01-012a
xor_gen_base	00-01-012b
pq_check_base	00-01-012c
xor_check_base	00-01-012d
ec_encode_data	00-06-0133
gf_vect_mul	00-05-0134
ec_encode_data_base	00-01-0135
gf_vect_mul_base	00-01-0136

Function	Version
gf_vect_dot_prod_base	00-01-0137
gf_vect_dot_prod	00-05-0138
gf_vect_dot_prod_avx2	04-05-0190
gf_2vect_dot_prod_avx	02-05-0191
gf_3vect_dot_prod_avx	02-05-0192
gf_4vect_dot_prod_avx	02-05-0193
gf_5vect_dot_prod_avx	02-04-0194
gf_6vect_dot_prod_avx	02-04-0195
gf_2vect_dot_prod_avx2	04-05-0196
gf_3vect_dot_prod_avx2	04-05-0197
gf_4vect_dot_prod_avx2	04-05-0198
gf_5vect_dot_prod_avx2	04-04-0199
gf_6vect_dot_prod_avx2	04-04-019a
gf_vect_mad_sse	00-01-0200
gf_vect_mad_avx	02-01-0201
gf_vect_mad_avx2	04-01-0202
gf_2vect_mad_sse	00-01-0203
gf_2vect_mad_avx	02-01-0204
gf_2vect_mad_avx2	04-01-0205
gf_3vect_mad_sse	00-01-0206
gf_3vect_mad_avx	02-01-0207
gf_3vect_mad_avx2	04-01-0208
gf_4vect_mad_sse	00-01-0209
gf_4vect_mad_avx	02-01-020a
gf_4vect_mad_avx2	04-01-020b
gf_5vect_mad_sse	00-01-020c
gf_5vect_mad_avx	02-01-020d
gf_5vect_mad_avx2	04-01-020e
gf_6vect_mad_sse	00-01-020f
gf_6vect_mad_avx	02-01-0210
gf_6vect_mad_avx2	04-01-0211
ec_encode_data_update	00-05-0212
gf_vect_mad	00-04-0213
gf_mul	00-00-0214
gf_invert_matrix	00-00-0215
gf_gen_rs_matrix	00-00-0216
gf_gen_cauchy1_matrix	00-00-0217

CHAPTER 3
INSTRUCTION SET REQUIREMENTS

crc16_t10dif_01 (uint16_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc16_t10dif_by4 (uint16_t init_crc, const unsigned char *buf, uint64_t len)
SSE4, PCLMULQDQ.

crc32_gzip_refl_by8 (uint32_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc32_ieee_01 (uint32_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc32_ieee_by4 (uint32_t init_crc, const unsigned char *buf, uint64_t len)
SSE4, PCLMULQDQ.

crc32_iscsi_00 (unsigned char *buffer, int len, unsigned int init_crc)
SSE4.2

crc32_iscsi_01 (unsigned char *buffer, int len, unsigned int init_crc)
SSE4.2, CLMUL

crc32_iscsi_baseline (unsigned char *buffer, int len, unsigned int init_crc)
SSE4.2

crc32_iscsi_simple (unsigned char *buffer, int len, unsigned int init_crc)
SSE4.2

crc64_ecma_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc64_ecma_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc64_iso_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc64_iso_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc64_jones_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

crc64_jones_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

ec_encode_data_avx (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)
AVX

ec_encode_data_avx2 (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)
AVX2

ec_encode_data_sse (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)
SSE4.1

ec_encode_data_update_avx (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)
AVX

ec_encode_data_update_avx2 (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)
AVX2

ec_encode_data_update_sse (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)
SSE4.1

gf_2vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX

gf_2vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX2

gf_2vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
SSE4.1

gf_2vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX

gf_2vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX2

gf_2vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
SSE4.1

gf_3vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX

gf_3vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX2

gf_3vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
SSE4.1

gf_3vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX

gf_3vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX2

gf_3vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
SSE4.1

gf_4vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX

gf_4vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX2

gf_4vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
SSE4.1

gf_4vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX

gf_4vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX2

gf_4vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
SSE4.1

gf_5vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX

gf_5vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX2

gf_5vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
SSE4.1

gf_5vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX

gf_5vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX2

gf_5vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
SSE4.1

gf_6vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX

gf_6vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
AVX2

gf_6vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)
SSE4.1

gf_6vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX

gf_6vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
AVX2

gf_6vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)
SSE4.1

gf_vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)
AVX

gf_vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)
AVX2

gf_vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)
SSE4.1

gf_vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)
AVX

gf_vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)
AVX2

gf_vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)
SSE4.1

gf_vect_mul_avx (int len, unsigned char *gftbl, void *src, void *dest)
AVX

gf_vect_mul_sse (int len, unsigned char *gftbl, void *src, void *dest)
SSE4.1

mem_cmp_avx (void *src, void *des, int n)
AVX

mem_cmp_avx2 (void *src, void *des, int n)
AVX2

mem_cmp_sse (void *src, void *des, int n)
SSE4.1

mem_cpy_avx (void *des, void *src, int n)
AVX

mem_cpy_sse (void *des, void *src, int n)
SSE2

mem_zero_detect_avx (void *mem, int len)
AVX

pq_check_sse (int vects, int len, void **array)
SSE4.1

pq_gen_avx (int vects, int len, void **array)
AVX

pq_gen_avx2 (int vects, int len, void **array)
AVX2

pq_gen_sse (int vects, int len, void **array)
SSE4.1

xor_check_sse (int vects, int len, void **array)
SSE4.1

xor_gen_avx (int vects, int len, void **array)
AVX

xor_gen_sse (int vects, int len, void **array)
SSE4.1

CHAPTER 4
DATA STRUCTURE INDEX

4.1 Data Structures

Here are the data structures with brief descriptions:

CHAPTER 5
FILE INDEX

5.1 File List

Here is a list of all documented files with brief descriptions:

CHAPTER 6
DATA STRUCTURE DOCUMENTATION

6.1 BitBuf2 Struct Reference

Holds Bit Buffer information.

`#include <igzip_lib.h>`

Data Fields

- uint64_t m_bits
  bits in the bit buffer
- uint32_t m_bit_count
  number of valid bits in the bit buffer
- uint8_t * m_out_buf
  current index of buffer to write to
- uint8_t * m_out_end
  end of buffer to write to
- uint8_t * m_out_start
  start of buffer to write to

6.1.1 Detailed Description

Holds Bit Buffer information.

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.2 inflate_huff_code_large Struct Reference

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.3 inflate_huff_code_small Struct Reference

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.4 inflate_state Struct Reference

Holds decompression state information.

```
#include <igzip_lib.h>
```

Data Fields

- uint8_t * next_out
  Next output Byte.
- uint32_t avail_out
  Number of bytes available at next_out.
- uint32_t total_out
  Total bytes written out so far.
- uint8_t * next_in
  Next input byte.
- uint64_t read_in
  Bits buffered to handle unaligned streams.
- uint32_t avail_in
  Number of bytes available at next_in.
- int32_t read_in_length
  Bits in read_in.
- struct inflate_huff_code_large lit_huff_code
  Structure for decoding lit/len symbols.
- struct inflate_huff_code_small dist_huff_code
  Structure for decoding dist symbols.
- enum isal_block_state block_state
  Current decompression state.
- uint32_t dict_length
  Length of dictionary used.
- uint32_t bfinal
  Flag identifying final block.
- uint32_t crc_flag
  Flag identifying whether to track of crc.

- uint32_t crc

  Contains crc of output if crc_flag is set.
- int32_t type0_block_len

  Length left to read of type 0 block when outbuffer overflow occured.
- int32_t copy_overflow_length

  Length left to copy when outbuffer overflow occured.
- int32_t copy_overflow_distance

  Lookback distance when outbuffer overlow occured.
- int32_t tmp_in_size

  Number of bytes in tmp_in_buffer.
- int32_t tmp_out_valid

  Number of bytes in tmp_out_buffer.
- int32_t tmp_out_processed

  Number of bytes processed in tmp_out_buffer.
- uint8_t tmp_in_buffer [ISAL_DEF_MAX_HDR_SIZE]

  Temporary buffer containing data from the input stream.
- uint8_t tmp_out_buffer [2 ∗ISAL_DEF_HIST_SIZE+ISAL_LOOK_AHEAD]

  Temporary buffer containing data from the output stream.

6.4.1 Detailed Description

Holds decompression state information.

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.5 isal_huff_histogram Struct Reference

Holds histogram of deflate symbols.

`#include <igzip_lib.h>`

Data Fields

- uint64_t lit_len_histogram [ISAL_DEF_LIT_LEN_SYMBOLS]

  Histogram of Literal/Len symbols seen.
- uint64_t dist_histogram [ISAL_DEF_DIST_SYMBOLS]

  Histogram of Distance Symbols seen.
- uint16_t hash_table [IGZIP_HASH_SIZE]

  Tmp space used as a hash table.

6.5.1 Detailed Description

Holds histogram of deflate symbols.

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.6 isal_hufftables Struct Reference

Holds the huffman tree used to huffman encode the input stream.

`#include <igzip_lib.h>`

Data Fields

- uint8_t deflate_hdr [ISAL_DEF_MAX_HDR_SIZE]
  deflate huffman tree header
- uint32_t deflate_hdr_count
  Number of whole bytes in deflate_huff_hdr.
- uint32_t deflate_hdr_extra_bits
  Number of bits in the partial byte in header.
- uint32_t dist_table [IGZIP_DIST_TABLE_SIZE]
  bits 4:0 are the code length, bits 31:5 are the code
- uint32_t len_table [IGZIP_LEN_TABLE_SIZE]
  bits 4:0 are the code length, bits 31:5 are the code
- uint16_t lit_table [IGZIP_LIT_TABLE_SIZE]
  literal code
- uint8_t lit_table_sizes [IGZIP_LIT_TABLE_SIZE]
  literal code length
- uint16_t dcodes [30-IGZIP_DECODE_OFFSET]
  distance code
- uint8_t dcodes_sizes [30-IGZIP_DECODE_OFFSET]
  distance code length

6.6.1 Detailed Description

Holds the huffman tree used to huffman encode the input stream.

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.7 isal_mod_hist Struct Reference

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.8 isal_zstate Struct Reference

Holds the internal state information for input and output compression streams.

`#include <igzip_lib.h>`

Data Fields

- uint32_t b_bytes_valid
  number of bytes of valid data in buffer
- uint32_t b_bytes_processed
  keeps track of the number of bytes processed in isal_zstate.buffer
- uint8_t * file_start
  pointer to where file would logically start
- uint32_t crc
  Current crc.
- struct BitBuf2 bitbuf
  Bit Buffer.
- enum isal_zstate_state state
  Current state in processing the data stream.
- uint32_t count
  used for partial header/trailer writes
- uint8_t tmp_out_buff [16]
  temporary array
- uint32_t tmp_out_start
  temporary variable
- uint32_t tmp_out_end
  temporary variable
- uint32_t has_wrap_hdr
  keeps track of wrapper header
- uint32_t has_eob
  keeps track of eob on the last deflate block
- uint32_t has_eob_hdr
  keeps track of eob hdr (with BFINAL set)

- uint32_t has_hist

  flag to track if there is match history
- ALIGN uint8_t buffer [2 ∗(32 ∗1024)+(18 ∗16)]

  Internal buffer.
- ALIGN uint16_t head [(8 ∗1024)]

  Hash array.

6.8.1 Detailed Description

Holds the internal state information for input and output compression streams.

The documentation for this struct was generated from the following file:

- igzip_lib.h

6.9 isal_zstream Struct Reference

Holds stream information.

`#include <igzip_lib.h>`

Data Fields

- uint8_t ∗ next_in

  Next input byte.
- uint32_t avail_in

  number of bytes available at next_in
- uint32_t total_in

  total number of bytes read so far
- uint8_t ∗ next_out

  Next output byte.
- uint32_t avail_out

  number of bytes available at next_out
- uint32_t total_out

  total number of bytes written so far
- struct isal_hufftables ∗ hufftables

  Huffman encoding used when compressing.
- uint32_t level

  Compression level to use.
- uint32_t level_buf_size

Size of level_buf.

- uint8_t * level_buf

  User allocated buffer required for different compression levels.

- uint32_t end_of_stream

  non-zero if this is the last input buffer

- uint32_t flush

  Flush type can be NO_FLUSH, SYNC_FLUSH or FULL_FLUSH.

- uint32_t gzip_flag

  Indicate if gzip compression is to be performed.

- struct isal_zstate internal_state

  Internal state for this stream.

6.9.1 Detailed Description

Holds stream information.

Examples:

igzip_example.c.

The documentation for this struct was generated from the following file:

- igzip_lib.h

CHAPTER 7
FILE DOCUMENTATION

7.1 crc.h File Reference

CRC functions.

```
#include <stdint.h>
```

Functions

- uint16_t crc16_t10dif (uint16_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the T10 standard, runs appropriate version.
- uint32_t crc32_ieee (uint32_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the IEEE standard, runs appropriate version.
- uint32_t crc32_gzip_refl (uint32_t init_crc, const unsigned char *buf, uint64_t len)

  Generate the customized CRC based on RFC 1952 CRC (http://www.ietf.org/rfc/rfc1952.txt) standard, runs appropriate version.
- unsigned int crc32_iscsi (unsigned char *buffer, int len, unsigned int init_crc)

  ISCSI CRC function, runs appropriate version.
- uint16_t crc16_t10dif_01 (uint16_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the T10 standard.
- uint16_t crc16_t10dif_by4 (uint16_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the T10 standard. Optimized for SLM.
- uint32_t crc32_ieee_01 (uint32_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the IEEE standard.
- uint32_t crc32_ieee_by4 (uint32_t init_crc, const unsigned char *buf, uint64_t len)

  Generate CRC from the IEEE standard.Optimized for SLM.
- uint32_t crc32_gzip_refl_by8 (uint32_t init_crc, const unsigned char *buf, uint64_t len)

  Generate the customized CRC based on RFC 1952 CRC (http://www.ietf.org/rfc/rfc1952.txt) standard.
- unsigned int crc32_iscsi_simple (unsigned char *buffer, int len, unsigned int init_crc)

  ISCSI CRC simple implementation with CRC32 instruction.
- unsigned int crc32_iscsi_baseline (unsigned char *buffer, int len, unsigned int init_crc)

  ISCSI CRC baseline implementation with CRC32 instruction.
- unsigned int crc32_iscsi_00 (unsigned char *buffer, int len, unsigned int init_crc)

  ISCSI CRC function optimized for Nehalem.
- unsigned int crc32_iscsi_01 (unsigned char *buffer, int len, unsigned int init_crc)

  ISCSI CRC function optimized for Westmere.

- unsigned int crc32_iscsi_base (unsigned char *buffer, int len, unsigned int crc_init)

  ISCSI CRC function, baseline version.
- uint16_t crc16_t10dif_base (uint16_t seed, uint8_t *buf, uint64_t len)

  Generate CRC from the T10 standard, runs baseline version.
- uint32_t crc32_ieee_base (uint32_t seed, uint8_t *buf, uint64_t len)

  Generate CRC from the IEEE standard, runs baseline version.
- uint32_t crc32_gzip_refl_base (uint32_t seed, uint8_t *buf, uint64_t len)

  *Generate the customized CRC based on RFC 1952 CRC (*http://www.ietf.org/rfc/rfc1952.txt*) standard, runs baseline version.*

7.1.1 Detailed Description

CRC functions.

7.1.2 Function Documentation

7.1.2.1 uint16_t crc16_t10dif (uint16_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the T10 standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

16 bit CRC

Parameters

init_crc	initial CRC value, 16 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

Examples:

crc_simple_test.c.

7.1.2.2 uint16_t crc16_t10dif_01 (uint16_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the T10 standard.

Requires SSE3, CLMUL

Returns

16 bit CRC

Parameters

init_crc	initial CRC value, 16 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.3 uint16_t crc16_t10dif_base (uint16_t *seed*, uint8_t * *buf*, uint64_t *len*)

Generate CRC from the T10 standard, runs baseline version.

Returns

16 bit CRC

Parameters

seed	initial CRC value, 16 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.4 uint16_t crc16_t10dif_by4 (uint16_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the T10 standard. Optimized for SLM.

Requires SSE4, PCLMULQDQ.

Returns

16 bit CRC

Parameters

init_crc	initial CRC value, 16 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.5 uint32_t crc32_gzip_refl (uint32_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate the customized CRC based on RFC 1952 CRC (http://www.ietf.org/rfc/rfc1952.txt) standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Note: CRC32 IEEE standard is widely used in HDLC, Ethernet, Gzip and many others. Its polynomial is 0x04C11DB7 in normal and 0xEDB88320 in reflection (or reverse). In ISA-L CRC, function crc32_ieee is actually designed for normal CRC32 IEEE version. And function crc32_gzip_refl is actually designed for reflected CRC32 IEEE. These two versions of CRC32 IEEE are not compatible with each other. Users who want to replace their not optimized crc32 ieee with ISA-L's crc32 function should be careful of that. Since many applications use CRC32 IEEE reflected version, Please have a check whether crc32_gzip_refl is right one for you instead of crc32_ieee.

Returns

32 bit CRC

Parameters

init_crc	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.6 uint32_t crc32_gzip_refl_base (uint32_t *seed*, uint8_t * *buf*, uint64_t *len*)

Generate the customized CRC based on RFC 1952 CRC (http://www.ietf.org/rfc/rfc1952.txt) standard, runs baseline version.

Returns

32 bit CRC

Parameters

seed	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.7 uint32_t crc32_gzip_refl_by8 (uint32_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate the customized CRC based on RFC 1952 CRC (http://www.ietf.org/rfc/rfc1952.txt) standard.

Requires SSE3, CLMUL

Returns

32 bit CRC

Parameters

init_crc	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.8 uint32_t crc32_ieee (uint32_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the IEEE standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime. Note: CRC32 IEEE standard is widely used in HDLC, Ethernet, Gzip and many others. Its polynomial is 0x04C11DB7 in normal and 0xEDB88320 in reflection (or reverse). In ISA-L CRC, function crc32_ieee is actually designed for normal CRC32 IEEE version. And function crc32_gzip_refl is actually designed for reflected CRC32 IEEE. These two versions of CRC32 IEEE are not compatible with each other. Users who want to replace their not optimized crc32 ieee with ISA-L's crc32 function should be careful of that. Since many applications use CRC32 IEEE reflected version, Please have a check whether crc32_gzip_refl is right one for you instead of crc32_ieee.

Returns

32 bit CRC

Parameters

init_crc	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

Examples:

crc_simple_test.c.

7.1.2.9 uint32_t crc32_ieee_01 (uint32_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the IEEE standard.

Requires SSE3, CLMUL

Returns

32 bit CRC

Parameters

init_crc	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.10 uint32_t crc32_ieee_base (uint32_t *seed*, uint8_t * *buf*, uint64_t *len*)

Generate CRC from the IEEE standard, runs baseline version.

Returns

32 bit CRC

Parameters

seed	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.11 uint32_t crc32_ieee_by4 (uint32_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from the IEEE standard.Optimized for SLM.

Requires SSE4, PCLMULQDQ.

Returns

32 bit CRC.

Parameters

init_crc	initial CRC value, 32 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.1.2.12 unsigned int crc32_iscsi (unsigned char * *buffer,* int *len,* unsigned int *init_crc*)

ISCSI CRC function, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
init_crc	initial CRC value

7.1.2.13 unsigned int crc32_iscsi_00 (unsigned char * *buffer,* int *len,* unsigned int *init_crc*)

ISCSI CRC function optimized for Nehalem.

Requires SSE4.2

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
init_crc	initial CRC value

7.1.2.14 unsigned int crc32_iscsi_01 (unsigned char * *buffer,* int *len,* unsigned int *init_crc*)

ISCSI CRC function optimized for Westmere.

Requires SSE4.2, CLMUL

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
init_crc	initial CRC value

7.1.2.15 unsigned int crc32_iscsi_base (unsigned char * *buffer*, int *len*, unsigned int *crc_init*)

ISCSI CRC function, baseline version.

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
crc_init	initial CRC value

7.1.2.16 unsigned int crc32_iscsi_baseline (unsigned char * *buffer*, int *len*, unsigned int *init_crc*)

ISCSI CRC baseline implementation with CRC32 instruction.

ISCSI CRC function using the CRC32 instruction in an unrolled loop.

Requires SSE4.2

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
init_crc	initial CRC value

7.1.2.17 unsigned int crc32_iscsi_simple (unsigned char * *buffer*, int *len*, unsigned int *init_crc*)

ISCSI CRC simple implementation with CRC32 instruction.

ISCSI CRC function that uses the CRC32 instruction in a simple, codesize efficient manner.

Requires SSE4.2

Returns

32 bit CRC

Parameters

buffer	buffer to calculate CRC on
len	buffer length in bytes
init_crc	initial CRC value

7.2 crc64.h File Reference

CRC64 functions.

```
#include <stdint.h>
```

Functions

- uint64_t crc64_ecma_refl (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in reflected format, runs appropriate version.
- uint64_t crc64_ecma_norm (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in normal format, runs appropriate version.
- uint64_t crc64_iso_refl (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in reflected format, runs appropriate version.
- uint64_t crc64_iso_norm (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in normal format, runs appropriate version.
- uint64_t crc64_jones_refl (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in reflected format, runs appropriate version.
- uint64_t crc64_jones_norm (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in normal format, runs appropriate version.
- uint64_t crc64_ecma_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in reflected format.
- uint64_t crc64_ecma_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in normal format.
- uint64_t crc64_ecma_refl_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in reflected format, runs baseline version.
- uint64_t crc64_ecma_norm_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ECMA-182 standard in normal format, runs baseline version.

- uint64_t crc64_iso_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in reflected format.
- uint64_t crc64_iso_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in normal format.
- uint64_t crc64_iso_refl_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in reflected format, runs baseline version.
- uint64_t crc64_iso_norm_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from ISO standard in normal format, runs baseline version.
- uint64_t crc64_jones_refl_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in reflected format.
- uint64_t crc64_jones_norm_by8 (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in normal format.
- uint64_t crc64_jones_refl_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in reflected format, runs baseline version.
- uint64_t crc64_jones_norm_base (uint64_t init_crc, const unsigned char *buf, uint64_t len)
  Generate CRC from "Jones" coefficients in normal format, runs baseline version.

7.2.1 Detailed Description

CRC64 functions.

7.2.2 Function Documentation

7.2.2.1 uint64_t crc64_ecma_norm (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.2 uint64_t crc64_ecma_norm_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.3 uint64_t crc64_ecma_norm_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.4 uint64_t crc64_ecma_refl (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.5 uint64_t crc64_ecma_refl_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.6 uint64_t crc64_ecma_refl_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ECMA-182 standard in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.7 uint64_t crc64_iso_norm (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.8 uint64_t crc64_iso_norm_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.9 uint64_t crc64_iso_norm_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.10 uint64_t crc64_iso_refl (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.11 uint64_t crc64_iso_refl_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.12 uint64_t crc64_iso_refl_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from ISO standard in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.13 uint64_t crc64_jones_norm (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.14 uint64_t crc64_jones_norm_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.15 uint64_t crc64_jones_norm_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.16 uint64_t crc64_jones_refl (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.17 uint64_t crc64_jones_refl_base (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.2.2.18 uint64_t crc64_jones_refl_by8 (uint64_t *init_crc*, const unsigned char * *buf*, uint64_t *len*)

Generate CRC from "Jones" coefficients in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

init_crc	initial CRC value, 64 bits
buf	buffer to calculate CRC on
len	buffer length in bytes (64-bit data)

7.3 erasure_code.h File Reference

Interface to functions supporting erasure code encode and decode.

```
#include "gf_vect_mul.h"
```

Functions

- void ec_init_tables (int k, int rows, unsigned char *a, unsigned char *gftbls)

  Initialize tables for fast Erasure Code encode and decode.
- void ec_encode_data (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

  Generate or decode erasure codes on blocks of data, runs appropriate version.
- void ec_encode_data_sse (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

  Generate or decode erasure codes on blocks of data.
- void ec_encode_data_avx (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

  Generate or decode erasure codes on blocks of data.
- void ec_encode_data_avx2 (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)

  Generate or decode erasure codes on blocks of data.
- void ec_encode_data_base (int len, int srcs, int dests, unsigned char *v, unsigned char **src, unsigned char **dest)

  Generate or decode erasure codes on blocks of data, runs baseline version.
- void ec_encode_data_update (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

  Generate update for encode or decode of erasure codes from single source, runs appropriate version.
- void ec_encode_data_update_sse (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

  Generate update for encode or decode of erasure codes from single source.
- void ec_encode_data_update_avx (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

  Generate update for encode or decode of erasure codes from single source.
- void ec_encode_data_update_avx2 (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)

  Generate update for encode or decode of erasure codes from single source.
- void ec_encode_data_update_base (int len, int k, int rows, int vec_i, unsigned char *v, unsigned char *data, unsigned char **dest)

  Generate update for encode or decode of erasure codes from single source.
- void gf_vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)

GF(2^8) vector dot product.

- void gf_vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)

  GF(2^8) vector dot product.
- void gf_vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)

  GF(2^8) vector dot product.
- void gf_2vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with two outputs.
- void gf_2vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with two outputs.
- void gf_2vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with two outputs.
- void gf_3vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with three outputs.
- void gf_3vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with three outputs.
- void gf_3vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with three outputs.
- void gf_4vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with four outputs.
- void gf_4vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with four outputs.
- void gf_4vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with four outputs.
- void gf_5vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with five outputs.
- void gf_5vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with five outputs.
- void gf_5vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with five outputs.

- void gf_6vect_dot_prod_sse (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with six outputs.
- void gf_6vect_dot_prod_avx (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with six outputs.
- void gf_6vect_dot_prod_avx2 (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char **dest)

  GF(2^8) vector dot product with six outputs.
- void gf_vect_dot_prod_base (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)

  GF(2^8) vector dot product, runs baseline version.
- void gf_vect_dot_prod (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)

  GF(2^8) vector dot product, runs appropriate version.
- void gf_vect_mad (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply accumulate, runs appropriate version.
- void gf_vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply accumulate, arch specific version.
- void gf_vect_mad_base (int len, int vec, int vec_i, unsigned char *v, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply accumulate, baseline version.
- void gf_2vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 2 accumulate. SSE version.
- void gf_2vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 2 accumulate. AVX version of gf_2vect_mad_sse().
- void gf_2vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 2 accumulate. AVX2 version of gf_2vect_mad_sse().
- void gf_3vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 3 accumulate. SSE version.
- void gf_3vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 3 accumulate. AVX version of gf_3vect_mad_sse().

- void gf_3vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 3 accumulate. AVX2 version of gf_3vect_mad_sse().
- void gf_4vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 4 accumulate. SSE version.
- void gf_4vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 4 accumulate. AVX version of gf_4vect_mad_sse().
- void gf_4vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 4 accumulate. AVX2 version of gf_4vect_mad_sse().
- void gf_5vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 5 accumulate. SSE version.
- void gf_5vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 5 accumulate. AVX version.
- void gf_5vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 5 accumulate. AVX2 version.
- void gf_6vect_mad_sse (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 6 accumulate. SSE version.
- void gf_6vect_mad_avx (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 6 accumulate. AVX version.
- void gf_6vect_mad_avx2 (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char **dest)

  GF(2^8) vector multiply with 6 accumulate. AVX2 version.
- unsigned char gf_mul (unsigned char a, unsigned char b)

  Single element GF(2^8) multiply.
- unsigned char gf_inv (unsigned char a)

  Single element GF(2^8) inverse.
- void gf_gen_rs_matrix (unsigned char *a, int m, int k)

  Generate a matrix of coefficients to be used for encoding.
- void gf_gen_cauchy1_matrix (unsigned char *a, int m, int k)

  Generate a Cauchy matrix of coefficients to be used for encoding.
- int gf_invert_matrix (unsigned char *in, unsigned char *out, const int n)

  Invert a matrix in GF(2^8)

7.3.1 Detailed Description

Interface to functions supporting erasure code encode and decode. This file defines the interface to optimized functions used in erasure codes. Encode and decode of erasures in GF(2^8) are made by calculating the dot product of the symbols (bytes in GF(2^8)) across a set of buffers and a set of coefficients. Values for the coefficients are determined by the type of erasure code. Using a general dot product means that any sequence of coefficients may be used including erasure codes based on random coefficients. Multiple versions of dot product are supplied to calculate 1-6 output vectors in one pass. Base GF multiply and divide functions can be sped up by defining GF_LARGE_TABLES at the expense of memory size.

7.3.2 Function Documentation

7.3.2.1 void ec_encode_data (int *len*, int *k*, int *rows*, unsigned char * *gftbls*, unsigned char ** *data*, unsigned char ** *coding*)

Generate or decode erasure codes on blocks of data, runs appropriate version.

Given a list of source data blocks, generate one or multiple blocks of encoded data as specified by a matrix of GF(2^8) coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each block of data (vector) of source or dest data.
k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32*k*rows
data	Array of pointers to source input buffers.
coding	Array of pointers to coded output buffers.

Returns

none

7.3.2.2 void ec_encode_data_avx (int *len*, int *k*, int *rows*, unsigned char * *gftbls*, unsigned char ** *data*, unsigned char ** *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires AVX

7.3.2.3 void ec_encode_data_avx2 (int *len*, int *k*, int *rows*, unsigned char * *gftbls*, unsigned char ** *data*, unsigned char ** *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires AVX2

7.3.2.4 void ec_encode_data_base (int *len*, int *srcs*, int *dests*, unsigned char * *v*, unsigned char ** *src*, unsigned char ** *dest*)

Generate or decode erasure codes on blocks of data, runs baseline version.

Baseline version of ec_encode_data() with same parameters.

7.3.2.5 void ec_encode_data_sse (int *len*, int *k*, int *rows*, unsigned char * *gftbls*, unsigned char ** *data*, unsigned char ** *coding*)

Generate or decode erasure codes on blocks of data.

Arch specific version of ec_encode_data() with same parameters.

Requires SSE4.1

7.3.2.6 void ec_encode_data_update (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char * *g_tbls*, unsigned char * *data*, unsigned char ** *coding*)

Generate update for encode or decode of erasure codes from single source, runs appropriate version.

Given one source data block, update one or multiple blocks of encoded data as specified by a matrix of GF(2^8) coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes from one input source at a time.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each block of data (vector) of source or dest data.
k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
vec_i	The vector index corresponding to the single input source.
g_tbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32*k*rows
data	Pointer to single input source used to update output parity.
coding	Array of pointers to coded output buffers.

Returns

none

7.3.2.7 void ec_encode_data_update_avx (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char * *g_tbls*, unsigned char * *data*, unsigned char ** *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires AVX

7.3.2.8 void ec_encode_data_update_avx2 (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char * *g_tbls*, unsigned char * *data*, unsigned char ** *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires AVX2

7.3.2.9 void ec_encode_data_update_base (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char * *v*, unsigned char * *data*, unsigned char ** *dest*)

Generate update for encode or decode of erasure codes from single source.

Baseline version of ec_encode_data_update().

7.3.2.10 void ec_encode_data_update_sse (int *len*, int *k*, int *rows*, int *vec_i*, unsigned char * *g_tbls*, unsigned char * *data*, unsigned char ** *coding*)

Generate update for encode or decode of erasure codes from single source.

Arch specific version of ec_encode_data_update() with same parameters.

Requires SSE4.1

7.3.2.11 void ec_init_tables (int *k*, int *rows*, unsigned char * *a*, unsigned char * *gftbls*)

Initialize tables for fast Erasure Code encode and decode.

Generates the expanded tables needed for fast encode or decode for erasure codes on blocks of data. 32bytes is generated for each input coefficient.

Parameters

k	The number of vector sources or rows in the generator matrix for coding.
rows	The number of output vectors to concurrently encode/decode.
a	Pointer to sets of arrays of input coefficients used to encode or decode data.
gftbls	Pointer to start of space for concatenated output tables generated from input coefficients. Must be of size 32∗k∗rows.

Returns

none

7.3.2.12 void gf_2vect_dot_prod_avx (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.13 void gf_2vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding

encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.14 void gf_2vect_dot_prod_sse (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with two outputs.

Vector dot product optimized to calculate two ouputs at a time. Does two GF(2^8) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2∗32∗vlen byte constant array based on the two sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 2∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.15 void gf_2vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 2 accumulate. AVX version of gf_2vect_mad_sse().

Requires AVX

7.3.2.16 void gf_2vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 2 accumulate. AVX2 version of gf_2vect_mad_sse().

Requires AVX2

7.3.2.17 void gf_2vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 2 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32*vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32*vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

7.3.2.18 void gf_3vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3*32*vlen byte constant array based on the three sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 3*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.19 void gf_3vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3*32*vlen byte constant array based on the three sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 3∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.20 void gf_3vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with three outputs.

Vector dot product optimized to calculate three ouputs at a time. Does three GF(2^8) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3∗32∗vlen byte constant array based on the three sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 3∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.21 void gf_3vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 3 accumulate. AVX version of gf_3vect_mad_sse().

Requires AVX

7.3.2.22 void gf_3vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 3 accumulate. AVX2 version of gf_3vect_mad_sse().

Requires AVX2

7.3.2.23 void gf_3vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 3 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32*vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32*vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

7.3.2.24 void gf_4vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4*32*vlen byte constant array based on the four sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 4∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.25 void gf_4vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4∗32∗vlen byte constant array based on the four sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 4∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.26 void gf_4vect_dot_prod_sse (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with four outputs.

Vector dot product optimized to calculate four ouputs at a time. Does four GF(2^8) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure

coding encode and decode. Function requires pre-calculation of a 4∗32∗vlen byte constant array based on the four sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 4∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.27 void gf_4vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. AVX version of gf_4vect_mad_sse().

Requires AVX

7.3.2.28 void gf_4vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. AVX2 version of gf_4vect_mad_sse().

Requires AVX2

7.3.2.29 void gf_4vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 4 accumulate. SSE version.

Does a GF(2^8) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Pointer to source input array.
dest	Array of pointers to destination input/outputs.

Returns

none

7.3.2.30 void gf_5vect_dot_prod_avx (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.31 void gf_5vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.32 void gf_5vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with five outputs.

Vector dot product optimized to calculate five ouputs at a time. Does five GF(2^8) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5∗32∗vlen byte constant array based on the five sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 5∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.33 void gf_5vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 5 accumulate. AVX version.

Requires AVX

7.3.2.34 void gf_5vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 5 accumulate. AVX2 version.

Requires AVX2

7.3.2.35 void gf_5vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char * *gftbls*, unsigned char * *src*, unsigned char ** *dest*)

GF(2^8) vector multiply with 5 accumulate. SSE version.

Requires SSE4.1

7.3.2.36 void gf_6vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char ** *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6*32*vlen byte constant array based on the six sets of input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 6∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.37 void gf_6vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6∗32∗vlen byte constant array based on the six sets of input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 6∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.38 void gf_6vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector dot product with six outputs.

Vector dot product optimized to calculate six ouputs at a time. Does six GF(2^8) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding

encode and decode. Function requires pre-calculation of a 6∗32∗vlen byte constant array based on the six sets of input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 6∗32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Array of pointers to destination data buffers.

Returns

none

7.3.2.39 void gf_6vect_mad_avx (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 6 accumulate. AVX version.

Requires AVX

7.3.2.40 void gf_6vect_mad_avx2 (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 6 accumulate. AVX2 version.

Requires AVX2

7.3.2.41 void gf_6vect_mad_sse (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗∗ *dest*)

GF(2^8) vector multiply with 6 accumulate. SSE version.

Requires SSE4.1

7.3.2.42 void gf_gen_cauchy1_matrix (unsigned char * a, int m, int k)

Generate a Cauchy matrix of coefficients to be used for encoding.

Cauchy matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as 1/(i + j) | i != j, i:{0,k-1} j:{k,m-1}. Any sub-matrix of a Cauchy matrix should be invertable.

Parameters

a	[mxk] array to hold coefficients
m	number of rows in matrix corresponding to srcs + parity.
k	number of columns in matrix corresponding to srcs.

Returns

none

7.3.2.43 void gf_gen_rs_matrix (unsigned char * a, int m, int k)

Generate a matrix of coefficients to be used for encoding.

Vandermonde matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as 2^{i*(j-k+1)} i:{0,k-1} j:{k,m-1}. Commonly used method for choosing coefficients in erasure encoding but does not guarantee invertable for every sub matrix. For large pairs of m and k it is possible to find cases where the decode matrix chosen from sources and parity is not invertable. Users may want to adjust for certain pairs m and k. If m and k satisfy one of the following inequalities, no adjustment is required:

k <= 3 k = 4, m <= 25 k = 5, m <= 10 k <= 21, m-k = 4 m - k <= 3.

Parameters

a	[mxk] array to hold coefficients
m	number of rows in matrix corresponding to srcs + parity.
k	number of columns in matrix corresponding to srcs.

Returns

none

7.3.2.44 unsigned char gf_inv (unsigned char a)

Single element GF(2^8) inverse.

Parameters

a	Input element

Returns

Field element b such that a x b = {1}

7.3.2.45 int gf_invert_matrix (unsigned char * *in*, unsigned char * *out*, const int *n*)

Invert a matrix in GF(2^8)

Parameters

in	input matrix
out	output matrix such that [in] x [out] = [I] - identity matrix
n	size of matrix [nxn]

Returns

0 successful, other fail on singular input matrix

7.3.2.46 unsigned char gf_mul (unsigned char *a*, unsigned char *b*)

Single element GF(2^8) multiply.

Parameters

a	Multiplicand a
b	Multiplicand b

Returns

Product of a and b in GF(2^8)

7.3.2.47 void gf_vect_dot_prod (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char * *dest*)

GF(2^8) vector dot product, runs appropriate version.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32*vlen byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.48 void gf_vect_dot_prod_avx (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char * *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires AVX

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.49 void gf_vect_dot_prod_avx2 (int *len*, int *vlen*, unsigned char * *gftbls*, unsigned char ** *src*, unsigned char * *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires AVX2

Parameters

len	Length of each vector in bytes. Must be >= 32.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.50 void gf_vect_dot_prod_base (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector dot product, runs baseline version.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients. Only elements 32∗CONST∗j + 1 of this array are used, where j = (0, 1, 2...) and CONST is the number of elements in the array of input coefficients. The elements used correspond to the original input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.51 void gf_vect_dot_prod_sse (int *len*, int *vlen*, unsigned char ∗ *gftbls*, unsigned char ∗∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector dot product.

Does a GF(2^8) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32∗vlen byte constant array based on the input coefficients.

Requires SSE4.1

Parameters

len	Length of each vector in bytes. Must be >= 16.
vlen	Number of vector sources.
gftbls	Pointer to 32∗vlen byte array of pre-calculated constants based on the array of input coefficients.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.52 void gf_vect_mad (int *len*, int *vec*, int *vec_i*, unsigned char ∗ *gftbls*, unsigned char ∗ *src*, unsigned char ∗ *dest*)

GF(2^8) vector multiply accumulate, runs appropriate version.

Does a GF(2^8) multiply across each byte of input source with expanded constant and add to destination array. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32∗vec byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of each vector in bytes. Must be >= 32.
vec	The number of vector sources or rows in the generator matrix for coding.
vec_i	The vector index corresponding to the single input source.
gftbls	Pointer to array of input tables generated from coding coefficients in ec_init_tables(). Must be of size 32∗vec.
src	Array of pointers to source inputs.
dest	Pointer to destination data array.

Returns

none

7.3.2.53 void gf_vect_mad_avx (int *len,* int *vec,* int *vec_i,* unsigned char * *gftbls,* unsigned char * *src,* unsigned char * *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires AVX

7.3.2.54 void gf_vect_mad_avx2 (int *len,* int *vec,* int *vec_i,* unsigned char * *gftbls,* unsigned char * *src,* unsigned char * *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires AVX2

7.3.2.55 void gf_vect_mad_base (int *len,* int *vec,* int *vec_i,* unsigned char * *v,* unsigned char * *src,* unsigned char * *dest*)

GF(2^8) vector multiply accumulate, baseline version.

Baseline version of gf_vect_mad() with same parameters.

7.3.2.56 void gf_vect_mad_sse (int *len,* int *vec,* int *vec_i,* unsigned char * *gftbls,* unsigned char * *src,* unsigned char * *dest*)

GF(2^8) vector multiply accumulate, arch specific version.

Arch specific version of gf_vect_mad() with same parameters.

Requires SSE4.1

7.4 gf_vect_mul.h File Reference

Interface to functions for vector (block) multiplication in GF(2^8).

Functions

- int gf_vect_mul_sse (int len, unsigned char *gftbl, void *src, void *dest)

  GF(2^8) vector multiply by constant.

- int gf_vect_mul_avx (int len, unsigned char *gftbl, void *src, void *dest)

  GF(2^8) vector multiply by constant.
- int gf_vect_mul (int len, unsigned char *gftbl, void *src, void *dest)

  GF(2^8) vector multiply by constant, runs appropriate version.
- void gf_vect_mul_init (unsigned char c, unsigned char *gftbl)

  Initialize 32-byte constant array for GF(2^8) vector multiply.
- void gf_vect_mul_base (int len, unsigned char *a, unsigned char *src, unsigned char *dest)

  GF(2^8) vector multiply by constant, runs baseline version.

7.4.1 Detailed Description

Interface to functions for vector (block) multiplication in GF(2^8). This file defines the interface to routines used in fast RAID rebuild and erasure codes.

7.4.2 Function Documentation

7.4.2.1 int gf_vect_mul (int *len*, unsigned char * *gftbl*, void * *src*, void * *dest*)

GF(2^8) vector multiply by constant, runs appropriate version.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

7.4.2.2 int gf_vect_mul_avx (int *len*, unsigned char * *gftbl*, void * *src*, void * *dest*)

GF(2^8) vector multiply by constant.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array

based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Requires AVX

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

7.4.2.3 void gf_vect_mul_base (int *len*, unsigned char * *a*, unsigned char * *src*, unsigned char * *dest*)

GF(2^8) vector multiply by constant, runs baseline version.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
a	Pointer to 32-byte array of pre-calculated constants based on C. only use 2nd element is used.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

7.4.2.4 void gf_vect_mul_init (unsigned char *c*, unsigned char * *gftbl*)

Initialize 32-byte constant array for GF(2^8) vector multiply.

Calculates array {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} } as required by other fast vector multiply functions.

Parameters

c	Constant input.
gftbl	Table output.

7.4.2.5 int gf_vect_mul_sse (int *len*, unsigned char * *gftbl*, void * *src*, void * *dest*)

GF(2^8) vector multiply by constant.

Does a GF(2^8) vector multiply b = Ca where a and b are arrays and C is a single field element in GF(2^8). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C. gftbl(C) = {C{00}, C{01}, C{02}, ... , C{0f} }, {C{00}, C{10}, C{20}, ... , C{f0} }. Len and src must be aligned to 32B.

Requires SSE4.1

Parameters

len	Length of vector in bytes. Must be aligned to 32B.
gftbl	Pointer to 32-byte array of pre-calculated constants based on C.
src	Pointer to src data array. Must be aligned to 32B.
dest	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

7.5 igzip_lib.h File Reference

This file defines the igzip compression and decompression interface, a high performance deflate compression interface for storage applications.

```
#include <stdint.h>
#include "types.h"
```

Data Structures

- struct isal_huff_histogram

  Holds histogram of deflate symbols.
- struct isal_mod_hist
- struct BitBuf2

  Holds Bit Buffer information.
- struct isal_zstate

  Holds the internal state information for input and output compression streams.
- struct isal_hufftables

  Holds the huffman tree used to huffman encode the input stream.
- struct isal_zstream

Holds stream information.

- struct inflate_huff_code_large
- struct inflate_huff_code_small
- struct inflate_state

  Holds decompression state information.

Enumerations

- enum isal_zstate_state {
  ZSTATE_NEW_HDR, ZSTATE_HDR, ZSTATE_CREATE_HDR, ZSTATE_BODY,
  ZSTATE_FLUSH_READ_BUFFER , ZSTATE_TYPE0_BODY, ZSTATE_SYNC_FLUSH, ZSTATE_FLUSH_WRITE_BUFFER,
  ZSTATE_TRL, ZSTATE_END, ZSTATE_TMP_NEW_HDR, ZSTATE_TMP_HDR,
  ZSTATE_TMP_CREATE_HDR, ZSTATE_TMP_BODY, ZSTATE_TMP_FLUSH_READ_BUFFER , ZSTATE_TMP_SYNC_FLUSH,
  ZSTATE_TMP_FLUSH_WRITE_BUFFER, ZSTATE_TMP_TRL, ZSTATE_TMP_END }

  Compression State please note ZSTATE_TRL only applies for GZIP compression.

Functions

- void isal_update_histogram (uint8_t *in_stream, int length, struct isal_huff_histogram *histogram)

  Updates histograms to include the symbols found in the input stream. Since this function only updates the histograms, it can be called on multiple streams to get a histogram better representing the desired data set. When first using histogram it must be initialized by zeroing the structure.
- int isal_create_hufftables (struct isal_hufftables *hufftables, struct isal_huff_histogram *histogram)

  Creates a custom huffman code for the given histograms in which every literal and repeat length is assigned a code and all possible lookback distances are assigned a code.
- int isal_create_hufftables_subset (struct isal_hufftables *hufftables, struct isal_huff_histogram *histogram)

  Creates a custom huffman code for the given histograms like isal_create_hufftables() except literals with 0 frequency in the histogram are not assigned a code.
- void isal_deflate_init (struct isal_zstream *stream)

  Initialize compression stream data structure.
- void isal_deflate_reset (struct isal_zstream *stream)

  Reinitialize compression stream data structure. Performs the same action as isal_deflate_init, but does not change user supplied input such as the level, flush type, compression wrapper (like gzip), hufftables, and end_of_stream_flag.
- int isal_deflate_set_hufftables (struct isal_zstream *stream, struct isal_hufftables *hufftables, int type)

  Set stream to use a new Huffman code.
- void isal_deflate_stateless_init (struct isal_zstream *stream)

  Initialize compression stream data structure.
- int isal_deflate_set_dict (struct isal_zstream *stream, uint8_t *dict, uint32_t dict_len)

  Set compression dictionary to use.
- int isal_deflate (struct isal_zstream *stream)

Fast data (deflate) compression for storage applications.

- int isal_deflate_stateless (struct isal_zstream *stream)

  Fast data (deflate) stateless compression for storage applications.
- void isal_inflate_init (struct inflate_state *state)

  Initialize decompression state data structure.
- void isal_inflate_reset (struct inflate_state *state)

  Reinitialize decompression state data structure.
- int isal_inflate_set_dict (struct inflate_state *state, uint8_t *dict, uint32_t dict_len)

  Set decompression dictionary to use.
- int isal_inflate (struct inflate_state *state)

  Fast data (deflate) decompression for storage applications.
- int isal_inflate_stateless (struct inflate_state *state)

  Fast data (deflate) stateless decompression for storage applications.

7.5.1 Detailed Description

This file defines the igzip compression and decompression interface, a high performance deflate compression interface for storage applications. Deflate is a widely used compression standard that can be used standalone, it also forms the basis of gzip and zlib compression formats. Igzip supports the following flush features:

- No Flush: The default method where no special flush is performed.
- Sync flush: whereby isal_deflate() finishes the current deflate block at the end of each input buffer. The deflate block is byte aligned by appending an empty stored block.
- Full flush: whereby isal_deflate() finishes and aligns the deflate block as in sync flush but also ensures that subsequent block's history does not look back beyond this point and new blocks are fully independent.

Igzip also supports compression levels from ISAL_DEF_MIN_LEVEL to ISAL_DEF_MAX_LEVEL.

Igzip contains some behaviour configurable at compile time. These configureable options are:

- IGZIP_HIST_SIZE - Defines the window size. The default value is 32K (note K represents 1024), but 8K is also supported. Powers of 2 which are at most 32K may also work.
- LONGER_HUFFTABLES - Defines whether to use a larger hufftables structure which may increase performance with smaller IGZIP_HIST_SIZE values. By default this optoin is not defined. This define sets IGZIP_HIST_SIZE to be 8 if IGZIP_HIST_SIZE > 8K.

  As an example, to compile gzip with an 8K window size, in a terminal run

  ```
  gmake D="-D IGZIP_HIST_SIZE=8*1024"
  ```

  on Linux and FreeBSD, or with

  ```
  nmake -f Makefile.nmake D="-D
  *   IGZIP_HIST_SIZE=8*1024"
  ```

  on Windows.

7.5.2 Enumeration Type Documentation

7.5.2.1 enum isal_zstate_state

Compression State please note ZSTATE_TRL only applies for GZIP compression.

Enumerator

ZSTATE_NEW_HDR Header to be written.

ZSTATE_HDR Header state.

ZSTATE_CREATE_HDR Header to be created.

ZSTATE_BODY Body state.

ZSTATE_FLUSH_READ_BUFFER Flush buffer.

ZSTATE_TYPE0_BODY Type0 block header to be written. Type0 block body to be written

ZSTATE_SYNC_FLUSH Write sync flush block.

ZSTATE_FLUSH_WRITE_BUFFER Flush bitbuf.

ZSTATE_TRL Trailer state.

ZSTATE_END End state.

ZSTATE_TMP_NEW_HDR Temporary Header to be written.

ZSTATE_TMP_HDR Temporary Header state.

ZSTATE_TMP_CREATE_HDR Temporary Header to be created state.

ZSTATE_TMP_BODY Temporary Body state.

ZSTATE_TMP_FLUSH_READ_BUFFER Flush buffer.

ZSTATE_TMP_SYNC_FLUSH Write sync flush block.

ZSTATE_TMP_FLUSH_WRITE_BUFFER Flush bitbuf.

ZSTATE_TMP_TRL Temporary Trailer state.

ZSTATE_TMP_END Temporary End state.

7.5.3 Function Documentation

7.5.3.1 int isal_create_hufftables (struct isal_hufftables * *hufftables*, struct isal_huff_histogram * *histogram*)

Creates a custom huffman code for the given histograms in which every literal and repeat length is assigned a code and all possible lookback distances are assigned a code.

Parameters

hufftables	the output structure containing the huffman code
histogram	histogram containing frequency of literal symbols, repeat lengths and lookback distances

Returns

Returns a non zero value if an invalid huffman code was created.

7.5.3.2 int isal_create_hufftables_subset (struct isal_hufftables * *hufftables*, struct isal_huff_histogram * *histogram*)

Creates a custom huffman code for the given histograms like isal_create_hufftables() except literals with 0 frequency in the histogram are not assigned a code.

Parameters

hufftables	the output structure containing the huffman code
histogram	histogram containing frequency of literal symbols, repeat lengths and lookback distances

Returns

Returns a non zero value if an invalid huffman code was created.

7.5.3.3 int isal_deflate (struct isal_zstream * *stream*)

Fast data (deflate) compression for storage applications.

The call to isal_deflate() will take data from the input buffer (updating next_in, avail_in and write a compressed stream to the output buffer (updating next_out and avail_out). The function returns when either the input buffer is empty or the output buffer is full.

On entry to isal_deflate(), next_in points to an input buffer and avail_in indicates the length of that buffer. Similarly next_out points to an empty output buffer and avail_out indicates the size of that buffer.

The fields total_in and total_out start at 0 and are updated by isal_deflate(). These reflect the total number of bytes read or written so far.

When the last input buffer is passed in, signaled by setting the end_of_stream, the routine will complete compression at the end of the input buffer, as long as the output buffer is big enough.

The compression level can be set by setting level to any value between ISAL_DEF_MIN_LEVEL and ISAL_DEF_MAX_LEVEL. When the compression level is ISAL_DEF_MIN_LEVEL, hufftables can be set to a table trained for the the specific data type being compressed to achieve better compression. When a higher compression level is desired, a larger generic memory buffer needs to be supplied by setting level_buf and level_buf_size to represent the chunk of memory. For level x, the suggest size for this buffer this buffer is ISAL_DEFL_LVLx_DEFAULT. The defines ISAL_DEFL_LVLx_MIN, ISAL_DEFL_LVLx_SMALL, ISAL_DEFL_LVLx_MEDIUM, ISAL_DEFL_LVLx_LARGE, and ISAL_DEFL_LVLx_EXTRA_LARGE are also provided as other suggested sizes.

The equivalent of the zlib FLUSH_SYNC operation is currently supported. Flush types can be NO_FLUSH, SYNC_FLUSH or FULL_FLUSH. Default flush type is NO_FLUSH. A SYNC_ OR FULL_ flush will byte align the deflate block by appending an empty stored block once all input has been compressed, including the buffered input. Checking

that the out_buffer is not empty or that internal_state.state = ZSTATE_NEW_HDR is sufficient to guarantee all input has been flushed. Additionally FULL_FLUSH will ensure look back history does not include previous blocks so new blocks are fully independent. Switching between flush types is supported.

If a compression dictionary is required, the dictionary can be set calling isal_deflate_set_dictionary before calling isal_deflate.

If the gzip_flag is set to IGZIP_GZIP, a generic gzip header and the gzip trailer are written around the deflate compressed data. If gzip_flag is set to IGZIP_GZIP_NO_HDR, then only the gzip trailer is written.

Parameters

stream	Structure holding state information on the compression streams.

Returns

COMP_OK (if everything is ok), INVALID_FLUSH (if an invalid FLUSH is selected), ISAL_INVALID_LEVEL (if an invalid compression level is selected).

Examples:

igzip_example.c.

7.5.3.4 void isal_deflate_init (struct isal_zstream * *stream*)

Initialize compression stream data structure.

Parameters

stream	Structure holding state information on the compression streams.

Returns

none

Examples:

igzip_example.c.

7.5.3.5 void isal_deflate_reset (struct isal_zstream * *stream*)

Reinitialize compression stream data structure. Performs the same action as isal_deflate_init, but does not change user supplied input such as the level, flush type, compression wrapper (like gzip), hufftables, and end_of_stream_flag.

Parameters

stream	Structure holding state information on the compression streams.

Returns

none

7.5.3.6 int isal_deflate_set_dict (struct isal_zstream * *stream*, uint8_t * *dict*, uint32_t *dict_len*)

Set compression dictionary to use.

This function is to be called after isal_deflate_init, or after completing a SYNC_FLUSH or FULL_FLUSH and before the next call do isal_deflate. If the dictionary is longer than IGZIP_HIST_SIZE, only the last IGZIP_HIST_SIZE bytes will be used.

Parameters

stream	Structure holding state information on the compression streams.
dict	Array containing dictionary to use.
dict_len	Lenth of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

7.5.3.7 int isal_deflate_set_hufftables (struct isal_zstream * *stream*, struct isal_hufftables * *hufftables*, int *type*)

Set stream to use a new Huffman code.

Sets the Huffman code to be used in compression before compression start or after the sucessful completion of a SYNC_FLUSH or FULL_FLUSH. If type has value IGZIP_HUFFTABLE_DEFAULT, the stream is set to use the default Huffman code. If type has value IGZIP_HUFFTABLE_STATIC, the stream is set to use the deflate standard static Huffman code, or if type has value IGZIP_HUFFTABLE_CUSTOM, the stream is set to sue the isal_hufftables structure input to isal_deflate_set_hufftables.

Parameters

stream	Structure holding state information on the compression stream.
hufftables	new huffman code to use if type is set to IGZIP_HUFFTABLE_CUSTOM.
type	Flag specifying what hufftable to use.

Returns

Returns INVALID_OPERATION if the stream was unmodified. This may be due to the stream being in a state where changing the huffman code is not allowed or an invalid input is provided.

7.5.3.8 int isal_deflate_stateless (struct isal_zstream * *stream*)

Fast data (deflate) stateless compression for storage applications.

Stateless (one shot) compression routine with a similar interface to isal_deflate() but operates on entire input buffer at one time. Parameter avail_out must be large enough to fit the entire compressed output. Max expansion is limited to the input size plus the header size of a stored/raw block.

When the compression level is set to 1, unlike in isal_deflate(), level_buf may be optionally set depending on what what permormance is desired.

For stateless the flush types NO_FLUSH and FULL_FLUSH are supported. FULL_FLUSH will byte align the output deflate block so additional blocks can be easily appended.

If the gzip_flag is set to IGZIP_GZIP, a generic gzip header and the gzip trailer are written around the deflate compressed data. If gzip_flag is set to IGZIP_GZIP_NO_HDR, then only the gzip trailer is written.

Parameters

stream	Structure holding state information on the compression streams.

Returns

COMP_OK (if everything is ok), INVALID_FLUSH (if an invalid FLUSH is selected), ISAL_INVALID_LEVEL (if an invalid compression level is selected), STATELESS_OVERFLOW (if output buffer will not fit output).

7.5.3.9 void isal_deflate_stateless_init (struct isal_zstream * *stream*)

Initialize compression stream data structure.

Parameters

stream	Structure holding state information on the compression streams.

Returns

none

7.5.3.10 int isal_inflate (struct inflate_state * *state*)

Fast data (deflate) decompression for storage applications.

On entry to isal_inflate(), next_in points to an input buffer and avail_in indicates the length of that buffer. Similarly next_out points to an empty output buffer and avail_out indicates the size of that buffer.

The field total_out starts at 0 and is updated by isal_inflate(). This reflects the total number of bytes written so far.

The call to isal_inflate() will take data from the input buffer (updating next_in, avail_in and write a decompressed stream to the output buffer (updating next_out and avail_out). The function returns when the input buffer is empty, the output buffer is full or invalid data is found. The current state of the decompression on exit can be read from state->block-state. If the crc_flag is set to ISAL_GZIP_NO_HDR the gzip crc of the output is stored in state->crc. Alternatively, if the crc_flag is set to ISAL_ZLIB_NO_HDR the adler32 of the output is stored in state->crc.

If a dictionary is required, a call to isal_inflate_set_dict will set the dictionary.

Parameters

state	Structure holding state information on the compression streams.

Returns

ISAL_DECOMP_OK (if everything is ok), ISAL_END_INPUT (if all input was decompressed), ISAL_OUT_OVERFLOW (if output buffer ran out of space), ISAL_INVALID_BLOCK, ISAL_INVALID_SYMBOL, ISAL_INVALID_LOOKBACK.

7.5.3.11 void isal_inflate_init (struct inflate_state * *state*)

Initialize decompression state data structure.

Parameters

state	Structure holding state information on the compression streams.

Returns

none

7.5.3.12 void isal_inflate_reset (struct inflate_state * *state*)

Reinitialize decompression state data structure.

Parameters

state	Structure holding state information on the compression streams.

Returns

none

7.5.3.13 int isal_inflate_set_dict (struct inflate_state * *state*, uint8_t * *dict*, uint32_t *dict_len*)

Set decompression dictionary to use.

This function is to be called after isal_inflate_init. If the dictionary is longer than IGZIP_HIST_SIZE, only the last IGZIP_HIST_SIZE bytes will be used.

Parameters

state	Structure holding state information on the decompression stream.
dict	Array containing dictionary to use.
dict_len	Lenth of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

7.5.3.14 int isal_inflate_stateless (struct inflate_state * *state*)

Fast data (deflate) stateless decompression for storage applications.

Stateless (one shot) decompression routine with a similar interface to isal_inflate() but operates on entire input buffer at one time. Parameter avail_out must be large enough to fit the entire decompressed output.

Parameters

state	Structure holding state information on the compression streams.

Returns

ISAL_DECOMP_OK (if everything is ok), ISAL_END_INPUT (if all input was decompressed), ISAL_OUT_OVERFLOW (if output buffer ran out of space), ISAL_INVALID_BLOCK, ISAL_INVALID_SYMBOL, ISAL_INVALID_LOOKBACK.

7.5.3.15 void isal_update_histogram (uint8_t * *in_stream*, int *length*, struct isal_huff_histogram * *histogram*)

Updates histograms to include the symbols found in the input stream. Since this function only updates the histograms, it can be called on multiple streams to get a histogram better representing the desired data set. When first using histogram it must be initialized by zeroing the structure.

Parameters

in_stream	Input stream of data.
length	The length of start_stream.
histogram	The returned histogram of lit/len/dist symbols.

7.6 mem_routines.h File Reference

Interface to storage mem operations.

Functions

- int mem_zero_detect_avx (void *mem, int len)

  Detect if a memory region is all zero.
- int mem_cmp_sse (void *src, void *des, int n)

  Compare two memory blocks.
- int mem_cmp_avx (void *src, void *des, int n)

  Compare two memory blocks.
- int mem_cmp_avx2 (void *src, void *des, int n)

  Compare two memory blocks.
- void * mem_cpy_sse (void *des, void *src, int n)

  Copy memory blocks from src to des. Source and destination addresses cannot overlap.
- void * mem_cpy_avx (void *des, void *src, int n)

  Copy memory blocks from src to des. Source and destination addresses cannot overlap.

7.6.1 Detailed Description

Interface to storage mem operations. Defines the interface for vector versions of common memory functions. Vector memory functions are beneficial in some cases to standard library calls but not in all situations. Users should select vector versions when it is known from special use or environmental conditions that they will likely benefit.

7.6.2 Function Documentation

7.6.2.1 int mem_cmp_avx (void * *src*, void * *des*, int *n*)

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 256 bytes

Requires AVX

Parameters

src	the first memory region
des	the second memory region
n	the length of each memory region in bytes

Returns

0 - the two memory blocks are exactly the same other - the blocks are not the same

7.6.2.2 int mem_cmp_avx2 (void * *src*, void * *des*, int *n*)

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 256 bytes

Requires AVX2

Parameters

src	the first memory region
des	the second memory region
n	the length of each memory region in bytes

Returns

0 - the two memory blocks are exactly the same other - the blocks are not the same

7.6.2.3 int mem_cmp_sse (void * *src*, void * *des*, int *n*)

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 128 bytes

Requires SSE4.1

Parameters

src	the first memory region
des	the second memory region
n	the length of each memory region in bytes

Returns

0 - the two memory blocks are exactly the same other - the blocks are not the same

7.6.2.4 void* mem_cpy_avx (void * *des*, void * *src*, int *n*)

Copy memory blocks from src to des. Source and destination addresses cannot overlap.

Memory copy function with optimizations for large blocks > 256 bytes

Requires AVX

Parameters

src	the source memory region to copy from
des	the destination memory region to copy into
n	the length of memory region in bytes

Returns

the start address of the destination memory region

7.6.2.5 void* mem_cpy_sse (void * *des*, void * *src*, int *n*)

Copy memory blocks from src to des. Source and destination addresses cannot overlap.

Memory copy function with optimizations for large blocks > 128 bytes

Requires SSE2

Parameters

src	the source memory region to copy from
des	the destination memory region to copy into
n	the length of memory region in bytes

Returns

the start address of the destination memory region

7.6.2.6 int mem_zero_detect_avx (void * *mem*, int *len*)

Detect if a memory region is all zero.

Zero detect function with optimizations for large blocks > 128 bytes

Requires AVX

Parameters

mem	Pointer to memory region to test
len	Length of region in bytes

Returns

0 - region is all zeros other - region has non zero bytes

7.7 raid.h File Reference

Interface to RAID functions - XOR and P+Q calculation.

Functions

- int xor_gen (int vects, int len, void **array)

  Generate XOR parity vector from N sources, runs appropriate version.
- int xor_check (int vects, int len, void **array)

  Checks that array has XOR parity sum of 0 across all vectors, runs appropriate version.
- int pq_gen (int vects, int len, void **array)

  Generate P+Q parity vectors from N sources, runs appropriate version.
- int pq_check (int vects, int len, void **array)

  Checks that array of N sources, P and Q are consistent across all vectors, runs appropriate version.
- int xor_gen_sse (int vects, int len, void **array)

  Generate XOR parity vector from N sources.
- int xor_gen_avx (int vects, int len, void **array)

  Generate XOR parity vector from N sources.
- int xor_check_sse (int vects, int len, void **array)

  Checks that array has XOR parity sum of 0 across all vectors.
- int pq_gen_sse (int vects, int len, void **array)

  Generate P+Q parity vectors from N sources.
- int pq_gen_avx (int vects, int len, void **array)

  Generate P+Q parity vectors from N sources.
- int pq_gen_avx2 (int vects, int len, void **array)

  Generate P+Q parity vectors from N sources.
- int pq_check_sse (int vects, int len, void **array)

Checks that array of N sources, P and Q are consistent across all vectors.

- int pq_gen_base (int vects, int len, void **array)

  Generate P+Q parity vectors from N sources, runs baseline version.

- int xor_gen_base (int vects, int len, void **array)

  Generate XOR parity vector from N sources, runs baseline version.

- int xor_check_base (int vects, int len, void **array)

  Checks that array has XOR parity sum of 0 across all vectors, runs baseline version.

- int pq_check_base (int vects, int len, void **array)

  Checks that array of N sources, P and Q are consistent across all vectors, runs baseline version.

7.7.1 Detailed Description

Interface to RAID functions - XOR and P+Q calculation. This file defines the interface to optimized XOR calculation (RAID5) or P+Q dual parity (RAID6). Operations are carried out on an array of pointers to sources and output arrays.

7.7.2 Function Documentation

7.7.2.1 int pq_check (int *vects*, int *len*, void ** *array*)

Checks that array of N sources, P and Q are consistent across all vectors, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

vects	Number of vectors in array including P&Q.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and P, Q. P and Q parity are assumed to be the last two pointers in the array. All pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.2 int pq_check_base (int *vects*, int *len*, void ** *array*)

Checks that array of N sources, P and Q are consistent across all vectors, runs baseline version.

Parameters

vects	Number of vectors in array including P&Q.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and P, Q. P and Q parity are assumed to be the last two pointers in the array. All pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.3 int pq_check_sse (int *vects*, int *len*, void ** *array*)

Checks that array of N sources, P and Q are consistent across all vectors.

Requires SSE4.1

Parameters

vects	Number of vectors in array including P&Q.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and P, Q. P and Q parity are assumed to be the last two pointers in the array. All pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.4 int pq_gen (int *vects*, int *len*, void ** *array*)

Generate P+Q parity vectors from N sources, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes. Must be 32B aligned.
array	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

7.7.2.5 int pq_gen_avx (int *vects*, int *len*, void ** *array*)

Generate P+Q parity vectors from N sources.

Requires AVX

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.6 int pq_gen_avx2 (int *vects*, int *len*, void ** *array*)

Generate P+Q parity vectors from N sources.

Requires AVX2

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes. Must be 32B aligned.
array	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

7.7.2.7 int pq_gen_base (int *vects*, int *len*, void ** *array*)

Generate P+Q parity vectors from N sources, runs baseline version.

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.8 int pq_gen_sse (int *vects*, int *len*, void ** *array*)

Generate P+Q parity vectors from N sources.

Requires SSE4.1

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes. Must be 16B aligned.
array	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.9 int xor_check (int *vects*, int *len*, void ** *array*)

Checks that array has XOR parity sum of 0 across all vectors, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

vects	Number of vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to vectors. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.10 int xor_check_base (int *vects*, int *len*, void ** *array*)

Checks that array has XOR parity sum of 0 across all vectors, runs baseline version.

Parameters

vects	Number of vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to vectors. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

7.7.2.11 int xor_check_sse (int *vects*, int *len*, void ** *array*)

Checks that array has XOR parity sum of 0 across all vectors.

Requires SSE4.1

Parameters

vects	Number of vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to vectors. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

Examples:

xor_example.c.

7.7.2.12 int xor_gen (int *vects*, int *len*, void ** *array*)

Generate XOR parity vector from N sources, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

7.7.2.13 int xor_gen_avx (int *vects*, int *len*, void ** *array*)

Generate XOR parity vector from N sources.

Requires AVX

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

7.7.2.14 int xor_gen_base (int *vects*, int *len*, void ** *array*)

Generate XOR parity vector from N sources, runs baseline version.

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

7.7.2.15 int xor_gen_sse (int *vects*, int *len*, void ** *array*)

Generate XOR parity vector from N sources.

Requires SSE4.1

Parameters

vects	Number of source+dest vectors in array.
len	Length of each vector in bytes.
array	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

Examples:

xor_example.c.

CHAPTER 8
EXAMPLE DOCUMENTATION

8.1 crc_simple_test.c

Example usage of crc multibinary functions.

```
/**********************************************************************
  Copyright(c) 2011-2013 Intel Corporation All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions
  are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in
      the documentation and/or other materials provided with the
      distribution.
    * Neither the name of Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**********************************************************************/
#include <stdio.h>
#include <stdint.h>
#include "crc.h"

const uint16_t init_crc_16 = 0x1234;
const uint16_t t10_dif_expected = 0x60b3;
const uint32_t init_crc_32 = 0x12345678;
const uint32_t ieee_expected = 0x2ceadbe3;

int main(void)
{
        unsigned char p_buf[48];
        uint16_t t10_dif_computed;
        uint32_t ieee_computed;
        int i;

        for (i = 0; i < 48; i++)
                p_buf[i] = i;

        t10_dif_computed = crc16_t10dif(init_crc_16, p_buf, 48);

        if (t10_dif_computed != t10_dif_expected)
                printf("WRONG CRC-16(T10 DIF) value\n");
        else
                printf("CORRECT CRC-16(T10 DIF) value\n");

        ieee_computed = crc32_ieee(init_crc_32, p_buf, 48);
```

```
        if (ieee_computed != ieee_expected)
                printf("WRONG CRC-32(IEEE) value\n");
        else
                printf("CORRECT CRC-32(IEEE) value\n");

        return 0;
}
```

8.2 igzip_example.c

Example simple application using fast_lz.

```
/**********************************************************************
  Copyright(c) 2011-2016 Intel Corporation All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions
  are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in
      the documentation and/or other materials provided with the
      distribution.
    * Neither the name of Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "igzip_lib.h"

#define BUF_SIZE 8192
#ifndef LEVEL
# define LEVEL 0
#else
# define LEVEL 1
#endif

struct isal_zstream stream;

int main(int argc, char *argv[])
{
        uint8_t inbuf[BUF_SIZE], outbuf[BUF_SIZE];
        FILE *in, *out;

        if (argc != 3) {
                fprintf(stderr, "Usage: igzip_example infile outfile\n");
                exit(0);
        }
```

```
        in = fopen(argv[1], "rb");
        if (!in) {
                fprintf(stderr, "Can't open %s for reading\n", argv[1]);
                exit(0);
        }
        out = fopen(argv[2], "wb");
        if (!out) {
                fprintf(stderr, "Can't open %s for writing\n", argv[2]);
                exit(0);
        }

        printf("igzip_example\nWindow Size: %d K\n", IGZIP_HIST_SIZE / 1024);
        fflush(0);

        isal_deflate_init(&stream);
        stream.end_of_stream = 0;
        stream.flush = NO_FLUSH;

        if (LEVEL == 1) {
                stream.level = 1;
                stream.level_buf = malloc(ISAL_DEF_LVL1_DEFAULT);
                stream.level_buf_size = ISAL_DEF_LVL1_DEFAULT;
                if (stream.level_buf == 0) {
                        printf("Failed to allocate level compression buffer\n");
                        exit(0);
                }
        }

        do {
                stream.avail_in = (uint32_t) fread(inbuf, 1, BUF_SIZE, in);
                stream.end_of_stream = feof(in) ? 1 : 0;
                stream.next_in = inbuf;
                do {
                        stream.avail_out = BUF_SIZE;
                        stream.next_out = outbuf;

                        isal_deflate(&stream);

                        fwrite(outbuf, 1, BUF_SIZE - stream.avail_out, out);
                } while (stream.avail_out == 0);

                assert(stream.avail_in == 0);
        } while (stream.internal_state.state != ZSTATE_END);

        fclose(out);
        fclose(in);

        printf("End of igzip_example\n\n");
        return 0;
}
```

8.3 xor_example.c

Example of XOR usage on multiple sources.

```
/**********************************************************************
  Copyright(c) 2011-2013 Intel Corporation All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions
  are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
```

```
      notice, this list of conditions and the following disclaimer in
      the documentation and/or other materials provided with the
      distribution.
    * Neither the name of Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include "raid.h"
#include "types.h"

#define TEST_SOURCES 16
#define TEST_LEN     16*1024

int main(int argc, char *argv[])
{
        int i, j, should_pass, should_fail;
        void *buffs[TEST_SOURCES + 1];

        printf("XOR example\n");
        for (i = 0; i < TEST_SOURCES + 1; i++) {
                void *buf;
                if (posix_memalign(&buf, 16, TEST_LEN)) {
                        printf("alloc error: Fail");
                        return 1;
                }
                buffs[i] = buf;
        }

        printf("Make random data\n");
        for (i = 0; i < TEST_SOURCES + 1; i++)
                for (j = 0; j < TEST_LEN; j++)
                        ((char *)buffs[i])[j] = rand();

        printf("Generate xor parity\n");
        xor_gen_sse(TEST_SOURCES + 1, TEST_LEN, buffs);

        printf("Check parity: ");
        should_pass = xor_check_sse(TEST_SOURCES + 1, TEST_LEN, buffs);
        printf("%s\n", should_pass == 0 ? "Pass" : "Fail");

        printf("Find corruption: ");
        ((char *)buffs[TEST_SOURCES / 2])[TEST_LEN / 2] ^= 1;   // flip one bit
        should_fail = xor_check_sse(TEST_SOURCES + 1, TEST_LEN, buffs);    //recheck
        printf("%s\n", should_fail != 0 ? "Pass" : "Fail");

        return 0;
}
```

INDEX