Intel Intelligent Storage Acceleration Library

2.30.0

Generated by Doxygen 1.9.8

Chapter 1

Intel(R) Intelligent Storage Acceleration Library

ISA-L is a collection of optimized low-level functions targeting storage applications. ISA-L includes:

- Erasure codes - Fast block Reed-Solomon type erasure codes for any encode/decode matrix in GF(2^8).
- CRC - Fast implementations of cyclic redundancy check. Six different polynomials supported.
  - iscsi32, ieee32, t10dif, ecma64, iso64, jones64.
- Raid - calculate and operate on XOR and P+Q parity found in common RAID implementations.
- Compression - Fast deflate-compatible data compression.
- De-compression - Fast inflate-compatible data compression.

Also see:

- `ISA-L for updates`.
- For crypto functions see `isa-l_crypto on github`.
- The `github wiki` including a list of `distros/ports` offering binary packages.
- ISA-L `mailing list`.
- Contributing.

1.1 Building ISA-L

1.1.1 Prerequisites

- Make: GNU 'make' or 'nmake' (Windows).
- Optional: Building with autotools requires autoconf/automake packages.

x86_64:

- Assembler: nasm v2.11.01 or later (nasm v2.13 or better suggested for building in AVX512 support) or yasm version 1.2.0 or later.
- Compiler: gcc, clang, icc or VC compiler.

aarch64:

- Assembler: gas v2.24 or later.
- Compiler: gcc v4.7 or later.

other:

- Compiler: Portable base functions are available that build with most C compilers.

1.1.2 Autotools

To build and install the library with autotools it is usually sufficient to run:

```
./autogen.sh
./configure
make
sudo make install
```

1.1.3 Makefile

To use a standard makefile run:

```
make -f Makefile.unx
```

1.1.4 Windows

On Windows use nmake to build dll and static lib:

```
nmake -f Makefile.nmake
```

or see details on setting up environment here.

1.1.5 Other make targets

Other targets include:

- `make check` : create and run tests
- `make tests` : create additional unit tests
- `make perfs` : create included performance tests
- `make ex` : build examples
- `make other` : build other utilities such as compression file tests
- `make doc` : build API manual

Chapter 2

Contributing to ISA-L

Everyone is welcome to contribute. Patches may be submitted using GitHub pull requests (PRs). All commits must be signed off by the developer (–signoff) which indicates that you agree to the Developer Certificate of Origin. Patch discussion will happen directly on the GitHub PR. Design pre-work and general discussion occurs on the mailing list. Anyone can provide feedback in either location and all discussion is welcome. Decisions on whether to merge patches will be handled by the maintainer.

2.1 License

ISA-L is licensed using a BSD 3-clause [license]. All code submitted to the project is required to carry that license.

2.2 Certificate of Origin

In order to get a clear contribution chain of trust we use the signed-off-by language used by the Linux kernel project.

2.3 Mailing List

Contributors and users are welcome to submit new request on our roadmap, submit patches, file issues, and ask questions on our mailing list.

2.4 Coding Style

The coding style for ISA-L C code roughly follows linux kernel guidelines. Use the included indent script to format C code.

```
./tools/iindent your_files.c
```

And use check format script before submitting.

```
./tools/check_format.sh
```

Chapter 3

v2.30 Intel Intelligent Storage Acceleration Library Release Notes

RELEASE NOTE CONTENTS

1. KNOWN ISSUES
2. FIXED ISSUES
3. CHANGE LOG & FEATURES ADDED

3.1 1. KNOWN ISSUES

- Perf tests do not run in Windows environment.
- 32-bit lib is not supported in Windows.

3.2 2. FIXED ISSUES

v2.30

- Intel CET support.
- Windows nasm support fix.

v2.28

- Fix documentation on gf_vect_mad(). Min length listed as 32 instead of required min 64 bytes.

v2.27

- Fix lack of install for pkg-config files

v2.26

- Fixes for sanitizer warnings.

v2.25

- Fix for nasm on Mac OS X/darwin.

v2.24

- Fix for crc32_iscsi(). Potential read-over for small buffer. For an input buffer length of less than 8 bytes and aligned to an 8 byte boundary, function could read past length. Previously had the possibility to cause a seg fault only for length 0 and invalid buffer passed. Calculated CRC is unchanged.
- Fix for compression/decompression of $>$ 4GB files. For streaming compression of extremely large files, the total_out parameter would wrap and could potentially flag an otherwise valid lookback distance as being invalid. Total_out is still 32bit for zlib compatibility. No inconsistent compressed buffers were generated by the issue.

v2.23

- Fix for histogram generation base function.
- Fix library build warnings on macOS.
- Fix igzip to use bsf instruction when tzcnt is not available.

v2.22

- Fix ISA-L builds for other architectures. Base function and examples sanitized for non-IA builds.
- Fix fuzz test script to work with llvm 6.0 builtin libFuzz.

v2.20

- Inflate total_out behavior corrected for in-progress decompression. Previously total_out represented the total bytes decompressed into the output buffer or temp internal buffer. This is changed to be only the bytes put into the output buffer.
- Fixed issue with isal_create_hufftables_subset. Affects semi-dynamic compression use case when explicitly creating hufftables from histogram. The _hufftables_subset function could fail to generate length symbols for any length that were never seen.

v2.19

- Fix erasure code test that violates rs matrix bounds.
- Fix 0 length file and looping errors in igzip_inflate_test.

v2.18

- Mac OS X/darwin systems no longer require the –target=darwin config option. The autoconf canonical build should detect.

v2.17

- Fix igzip using 32K window and a shared object
- Fix igzip undefined instruction error on Nehalem.
- Fixed issue in crc performance tests where OS optimizations turned cold cache tests into warm tests.

v2.15

- Fix for windows register save in gf_6vect_mad_avx2.asm. Only affects windows versions of ec_encode_data_update() running with AVX2. A GP register was not properly restored resulting in corruption on return.

v2.14

- Building in unit directories is no longer supported removing the issue of leftover object files causing the top-level make build to fail.

v2.10

- Fix for windows register save overlap in gf_{3-6}vect_dot_prod_sse.asm. Only affects windows versions of erasure code. GP register saves/restore were pushed to same stack area as XMM.

3.3 3. CHANGE LOG & FEATURES ADDED

v2.30

- Igzip compression enhancements.
  - New functions for dictionary acceleration. Split dictionary processing and resetting can greatly accelerate the performance of compressing many small files with a dictionary.
  - New static level 0 header decode tables. Accelerates decompressing small files that are level 0 compressed by skipping the known header parsing.
  - New feature for igzip cli tool: support for concatenated .gz files. On decompression, igzip will process a series of independent, concatenated .gz files into one output stream.

- CRC Improvements
  - New vclmul version of crc32_iscsi().
  - Updates for aarch64.

v2.29

- CRC Improvements
  - New AVX512 vclmul versions of crc16_t10dif(), crc32_ieee(), crc32_gzip_refl.
- Erasure code improvements
  - Added AVX512 ec functions with 5 and 6 outputs. Can improve performance for codes with 5 or more parity by running in batches of up to 6 at a time.

v2.28

- New next-arch versions of 64-bit CRC. All norm and reflected 64-bit polynomials are expanded to utilize vpclmulqdq.

v2.27

- New multi-threaded compression option for igzip cli tool

v2.26

- Adler32 added to external API.
- Multi-arch improvements.
- Performance test improvements.

v2.25

- Igzip performance improvements and features.
  - Performance improvements for uncompressable files. Random or uncompressable files can be up to 3x faster in level 1 or 2 compression.
  - Additional small file performance improvments.
  - New options in igzip cli: use name from header or not, test compressed file.
- Multi-arch autoconf script.
  - Autoconf should detect architecture and run base functions at minimum.

v2.24

- Igzip small file performance improvements and new features.
  - Better performance on small files.
  - New gzip/zlib header and trailer handling.
  - New gzip/zlib header parsing helper functions.
  - New user-space compression/decompression tool igzip.
- New mem unit added with first function isal_zero_detect().

v2.23

- Igzip inflate (decompression) performance improvements.
  - Implemented multi-byte decode for inflate. Decode can pack up to three symbols into the decode table making some compressed streams decompress much faster depending on the prevalence of short codes.

v2.22

- Igzip: AVX2 version of level 3 compression added.
- Erasure code examples
  - New examples for standard EC encode and decode.
  - Example of piggyback EC encode and decode.

v2.21

- Igzip improvements
  - New compression levels added. ISA-L fast deflate now has more levels to balance speed vs. target compression level. Level 0, 1 are as in previous generations. New levels 2 & 3 target higher compression roughly comparable to zlib levels 2-3. Level 3 is currently only optimized for processors with AVX512 instructions.
- New T10dif & copy function - crc16_t10dif_copy()
  - CRC and copy was added to emulate T10dif operations such as DIF insert and strip. This function stitches together CRC and memcpy operations eliminating an extra data read.
- CRC32 iscsi performance improvements
  - Fixes issue under some distributions where warm cache performance was reduced.

v2.20

- Igzip improvements
  - Optimized deflate_hash in compression functions. Improves performance of using preset dictionary.
  - Removed alignment restrictions on input structure.

v2.19

- Igzip improvements
  - Add optimized Adler-32 checksum.
  - Implement zlib compression format.
  - Add stateful dictionary support.
  - Add struct reset functions for both deflate and inflate.
- Reflected IEEE format CRC32 is released out. Function interface is named crc32_gzip_refl.
- Exact work condition of Erasure Code Reed-Solomon Matrix is determined by new added program gen_rs_↩ matrix_limits.

v2.18

- New 2-pass fully-dynamic deflate compression (level -1). ISA-L fast deflate now has two levels. Level 0 (default) is the same as previous generations. Setting to level 1 will switch to the fully-dynamic compression that will typically reach higher compression ratios.
- RAID AVX512 functions.

v2.17

- New fast decompression (inflate)
- Compression improvements (deflate)
  - Speed and compression ratio improvements.
  - Fast custom Huffman code generation.
  - New features:
    - Run-time option of gzip crc calculation and headers/trailer.
    - Choice of static header (BTYPE 01) blocks.
    - LARGE_WINDOW, 32K history, now default.
    - Stateless full flush mode.
- CRC64
  - Six new 64-bit polynomials supported. Normal and reflected versions of ECMA, ISO and Jones polynomials.

v2.16

- Units added: crc, raid, igzip (deflate compression).

v2.15

- Erasure code updates. New AVX512 versions.
- Nasm support. ISA-L ported to build with nasm or yasm assembler.
- Windows DLL support. Windows builds DLL by default.

v2.14

- Autoconf and autotools build allows easier porting to additional systems. Previous make system still available to embedded users with Makefile.unx.
- Includes update for building on Mac OS X/darwin systems. Add –target=darwin to ./configure step.

v2.13

- Erasure code improvments
  - 32-bit port of optimized gf_vect_dot_prod() functions. This makes ec_encode_data() functions much faster on 32-bit processors.
  - Avoton performance improvements. Performance on Avoton for gf_vect_dot_prod() and ec_encode_data() can improve by as much as 20%.

v2.11

- Incremental erasure code. New functions added to erasure code to handle single source update of code blocks. The function ec_encode_data_update() works with parameters similar to ec_encode_data() but are called incrementally with each source block. These versions are useful when source blocks are not all available at once.

v2.10

- Erasure code updates
  - New AVX and AVX2 support functions.
  - Changes min len requirement on gf_vect_dot_prod() to 32 from 16.
  - Tests include both source and parity recovery with ec_encode_data().
  - New encoding examples with Vandermonde or Cauchy matrix.

v2.8

- First open release of erasure code unit that is part of ISA-L.

Chapter 4

ISA-L Testing

Tests are divided into check tests, unit tests and fuzz tests. Check tests, built with `make check`, should have no additional dependencies. Other unit tests built with `make test` may have additional dependencies in order to make comparisons of the output of ISA-L to other standard libraries and ensure compatibility. Fuzz tests are meant to be run with a fuzzing tool such as `AFL` or `llvm libFuzzer` fuzzing to direct the input data based on coverage. There are a number of scripts in the /tools directory to help with automating the running of tests.

4.1 Test check

`./tools/test_autorun.sh` is a helper script for kicking off check tests, that typically run for a few minutes, or extended tests that could run much longer. The command `test_autorun.sh check` build and runs all check tests with autotools and runs other short tests to ensure check tests, unit tests, examples, install, exe stack, format are correct. Each run of `test_autorun.sh` builds tests with a new random test seed that ensures that each run is unique to the seed but deterministic for debugging. Tests are also built with sanitizers and Electric Fence if available.

4.2 Extended tests

Extended tests are initiated with the command `./tools/test_autorun.sh ext`. These build and run check tests, unit tests, and other utilities that can take much longer than check tests alone. This includes special compression tools and some cross targets such as the no-arch build of base functions only and mingw build if tools are available.

4.3 Fuzz testing

`./tools/test_fuzz.sh` is a helper script for fuzzing to setup, build and run the ISA-L inflate fuzz tests on multiple fuzz tools. Fuzzing with `llvm libFuzzer` requires clang compiler tools with `-fsanitize=fuzzer` or `lib↩Fuzzer` installed. You can invoke the default fuzz tests under llvm with

```
./tools/test_fuzz.sh -e checked
```

To use `AFL`, install tools and system setup for `afl-fuzz` and run

```
./tools/test_fuzz.sh -e checked --afl 1 --llvm -1 -d 1
```

This uses internal vectors as a seed. You can also specify a sample file to use as a seed instead with `-f <file>`. One of three fuzz tests can be invoked: checked, simple, and round_trip.

Chapter 5

ISA-L Build Details

For x86-64 builds it is highly recommended to get an up-to-date version of `nasm` that can understand the latest instruction sets. Building with an older version is usually possible but the library may lack some function versions for the best performance.

5.1 Windows Build Environment Details

The windows dynamic and static libraries can be built with the nmake tool on the windows command line when appropriate paths and tools are setup as follows.

5.1.1 Download nasm and put into path

Download and install `nasm` and add location to path.

```
set PATH=%PATH%;C:\Program Files\NASM
```

5.1.2 Setup compiler environment

Install compiler and run environment setup script.

Compilers for windows usually have a batch file to setup environment variables for the command line called `vcvarsall.bat` or `compilervars.bat` or a link to run these. For Visual Studio this may be as follows for Community edition.

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat x64
```

For the Intel compiler the path is typically as follows where yyyy, x, zzz represent the version.

```
C:\Program Files (x86)\IntelSWTools\system_studio_for_windows_yyyy.x.zzz\compilers_and_libraries_yyyy\bin\compile
```

5.1.3 Build ISA-L libs and copy to appropriate place

Run `nmake /f Makefile.nmake`

This should build isa-l.dll, isa-l.lib and isa-l_static.lib. You may want to copy the libs to a system directory in the dynamic linking path such as `C:\windows\system32` or to a project directory.

To build a simple program with a static library.

```
cl /Fe: test.exe test.c isa-l_static.lib
```

Chapter 6

Instruction Set Requirements for arch-specific functions (non-multibinary)

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

SSE3, CLMUL

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

SSE3, CLMUL

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

SSE3, CLMUL

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

SSE3, CLMUL

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

SSE3, CLMUL

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

SSE3, CLMUL

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 8

File Index

8.1 File List

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

Chapter 9

Data Structure Documentation

9.1 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

9.1.1 Detailed Description

Holds Bit Buffer information.

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

- igzip_lib.h

9.2 inflate_huff_code_large Struct Reference

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

- igzip_lib.h

9.3 inflate_huff_code_small Struct Reference

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

- igzip_lib.h

9.4 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 or adler32 of output if crc_flag is set.
- uint32_t **hist_bits**

  Log base 2 of maximum lookback distance.
- int32_t **copy_overflow_length**

  Length left to copy when outbuffer overflow occurred.
- int32_t **copy_overflow_distance**

  Lookback distance when outbuffer overflow occurred.
- int16_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.

9.4.1 Detailed Description

Holds decompression state information.

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

- igzip_lib.h

9.5 isal_dict Struct Reference

Structure for holding processed dictionary information.

`#include <igzip_lib.h>`

9.5.1 Detailed Description

Structure for holding processed dictionary information.

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

- igzip_lib.h

9.6 isal_gzip_header Struct Reference

Data Fields

- uint32_t **text**

  Optional Text hint.
- uint32_t **time**

  Unix modification time in gzip header.
- uint32_t **xflags**

  xflags in gzip header
- uint32_t **os**

  OS in gzip header.
- uint8_t * **extra**

  Extra field in gzip header.
- uint32_t **extra_buf_len**

  Length of extra buffer.
- uint32_t **extra_len**

  Actual length of gzip header extra field.
- char * **name**

  Name in gzip header.
- uint32_t **name_buf_len**

  Length of name buffer.
- char * **comment**

  Comments in gzip header.
- uint32_t **comment_buf_len**

  Length of comment buffer.
- uint32_t **hcrc**

  Header crc or header crc flag.
- uint32_t **flags**

  Internal data.

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

- igzip_lib.h

9.7 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_LVL0_HASH_SIZE]
  Tmp space used as a hash table.

9.7.1 Detailed Description

Holds histogram of deflate symbols.

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

- igzip_lib.h

9.8 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

9.8.1 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

9.9 isal_mod_hist Struct Reference

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

- igzip_lib.h

9.10 isal_zlib_header Struct Reference

Data Fields

- uint32_t **info**

  base-2 logarithm of the LZ77 window size minus 8
- uint32_t **level**

  Compression level (fastest, fast, default, maximum)
- uint32_t **dict_id**

  Dictionary id.
- uint32_t **dict_flag**

  Whether to use a dictionary.

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

- igzip_lib.h

9.11 isal_zstate Struct Reference

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

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

Data Fields

- uint32_t **total_in_start**

  Not used, may be replaced with something else.
- uint32_t **block_next**

  Start of current deflate block in the input.
- uint32_t **block_end**

  End of current deflate block in the input.
- uint32_t **dist_mask**

  Distance mask used.
- enum isal_zstate_state **state**

  Current state in processing the data stream.
- struct BitBuf2 **bitbuf**

  Bit Buffer.
- uint32_t **crc**

  Current checksum without finalize step if any (adler)
- uint8_t **has_wrap_hdr**

  keeps track of wrapper header
- uint8_t **has_eob_hdr**

  keeps track of eob hdr (with BFINAL set)
- uint8_t **has_eob**

  keeps track of eob on the last deflate block
- uint8_t **has_hist**

  flag to track if there is match history
- uint16_t **has_level_buf_init**

  flag to track if user supplied memory has been initialized.
- 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 **b_bytes_valid**

  number of valid bytes in buffer
- uint32_t **b_bytes_processed**

  number of bytes processed in buffer
- uint8_t **buffer** [2 ∗IGZIP_HIST_SIZE+ISAL_LOOK_AHEAD]

  Internal buffer.
- uint16_t **head** [IGZIP_LVL0_HASH_SIZE]

  Hash array.

9.11.1 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

9.12 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.
- uint16_t **end_of_stream**
  non-zero if this is the last input buffer
- uint16_t **flush**
  Flush type can be NO_FLUSH, SYNC_FLUSH or FULL_FLUSH.
- uint16_t **gzip_flag**
  Indicate if gzip compression is to be performed.
- uint16_t **hist_bits**
  Log base 2 of maximum lookback distance, 0 is use default.
- struct isal_zstate **internal_state**
  Internal state for this stream.

9.12.1 Detailed Description

Holds stream information.

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

- igzip_lib.h

Chapter 10

File Documentation

10.1 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.
- uint16_t crc16_t10dif_copy (uint16_t init_crc, uint8_t *dst, uint8_t *src, uint64_t len)

  Generate CRC and copy 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.
- 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.
- uint16_t crc16_t10dif_copy_base (uint16_t init_crc, uint8_t *dst, uint8_t *src, uint64_t len)

  Generate CRC and copy 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.

10.1.1 Detailed Description

CRC functions.

10.1.2 Function Documentation

10.1.2.1 crc16_t10dif()

```
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)

10.1.2.2 crc16_t10dif_base()

```
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)

10.1.2.3 crc16_t10dif_copy()

```
uint16_t crc16_t10dif_copy (
            uint16_t init_crc,
            uint8_t * dst,
            uint8_t * src,
            uint64_t len )
```

Generate CRC and copy T10 standard, runs appropriate version.

Stitched CRC + copy function.

Returns

16 bit CRC

Parameters

init_crc	initial CRC value, 16 bits
dst	buffer destination for copy
src	buffer source to crc + copy
len	buffer length in bytes (64-bit data)

10.1.2.4 crc16_t10dif_copy_base()

```
uint16_t crc16_t10dif_copy_base (
            uint16_t init_crc,
            uint8_t * dst,
            uint8_t * src,
            uint64_t len )
```

Generate CRC and copy T10 standard, runs baseline version.

Returns

16 bit CRC

Parameters

init_crc	initial CRC value, 16 bits
dst	buffer destination for copy
src	buffer source to crc + copy
len	buffer length in bytes (64-bit data)

10.1.2.5 crc32_gzip_refl()

```
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)

10.1.2.6 crc32_gzip_refl_base()

```
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)

10.1.2.7 crc32_ieee()

```
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)

10.1.2.8 crc32_ieee_base()

```
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)

10.1.2.9 crc32_iscsi()

```
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

10.1.2.10 crc32_iscsi_base()

```
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

10.2 crc.h

Go to the documentation of this file.

```
00001 /*****************************************************************************
```

```
00002   Copyright(c) 2011-2015 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030
00037 #ifndef _CRC_H_
00038 #define _CRC_H_
00039
00040 #include <stdint.h>
00041
00042 #ifdef __cplusplus
00043 extern "C" {
00044 #endif
00045
00046
00047 /* Multi-binary functions */
00048
00057 uint16_t crc16_t10dif(
00058         uint16_t init_crc,
00059         const unsigned char *buf,
00060         uint64_t len
00061         );
00062
00063
00071 uint16_t crc16_t10dif_copy(
00072         uint16_t init_crc,
00073         uint8_t *dst,
00074         uint8_t *src,
00075         uint64_t len
00076         );
00077
00078
00099 uint32_t crc32_ieee(
00100         uint32_t init_crc,
00101         const unsigned char *buf,
00102         uint64_t len
00103         );
00104
00127 uint32_t crc32_gzip_refl(
00128         uint32_t init_crc,
00129         const unsigned char *buf,
00130         uint64_t len
00131         );
00132
00133
00142 unsigned int crc32_iscsi(
00143         unsigned char *buffer,
00144         int len,
00145         unsigned int init_crc
00146         );
00147
00148
00149 /* Base functions */
00150
00155 unsigned int crc32_iscsi_base(
00156         unsigned char *buffer,
00157         int len,
```

```
00158         unsigned int crc_init
00159         );
00160
00161
00166 uint16_t crc16_t10dif_base(
00167         uint16_t seed,
00168         uint8_t *buf,
00169         uint64_t len
00170         );
00171
00172
00177 uint16_t crc16_t10dif_copy_base(
00178         uint16_t init_crc,
00179         uint8_t *dst,
00180         uint8_t *src,
00181         uint64_t len
00182         );
00183
00184
00189 uint32_t crc32_ieee_base(
00190         uint32_t seed,
00191         uint8_t *buf,
00192         uint64_t len
00193         );
00194
00201 uint32_t crc32_gzip_refl_base(
00202         uint32_t seed,
00203         uint8_t *buf,
00204         uint64_t len
00205         );
00206
00207
00208 #ifdef __cplusplus
00209 }
00210 #endif
00211
00212 #endif // _CRC_H_
```

10.3 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.

10.3.1 Detailed Description

CRC64 functions.

10.3.2 Function Documentation

10.3.2.1 crc64_ecma_norm()

```
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)

10.3.2.2 crc64_ecma_norm_base()

```
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)

10.3.2.3 crc64_ecma_norm_by8()

```
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)

10.3.2.4 crc64_ecma_refl()

```
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)

10.3.2.5 crc64_ecma_refl_base()

```
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)

10.3.2.6 crc64_ecma_refl_by8()

```
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)

10.3.2.7 crc64_iso_norm()

```
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)

10.3.2.8 crc64_iso_norm_base()

```
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)

10.3.2.9 crc64_iso_norm_by8()

```
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)

10.3.2.10 crc64_iso_refl()

```
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)

10.3.2.11 crc64_iso_refl_base()

```
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)

10.3.2.12 crc64_iso_refl_by8()

```
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)

10.3.2.13 crc64_jones_norm()

```
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)

10.3.2.14 crc64_jones_norm_base()

```
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)

10.3.2.15 crc64_jones_norm_by8()

```
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)

10.3.2.16 crc64_jones_refl()

```
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)

10.3.2.17 crc64_jones_refl_base()

```
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)

10.3.2.18 crc64_jones_refl_by8()

```
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)

10.4 crc64.h

Go to the documentation of this file.

```
00001 /**********************************************************************
00002   Copyright(c) 2011-2016 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030
00037 #ifndef _CRC64_H_
00038 #define _CRC64_H_
00039
00040 #include <stdint.h>
00041
00042 #ifdef __cplusplus
00043 extern "C" {
00044 #endif
00045
00046
00047 /* Multi-binary functions */
00048
00057 uint64_t crc64_ecma_refl(
00058         uint64_t init_crc,
00059         const unsigned char *buf,
00060         uint64_t len
00061         );
00062
00071 uint64_t crc64_ecma_norm(
00072         uint64_t init_crc,
00073         const unsigned char *buf,
00074         uint64_t len
00075         );
00076
00085 uint64_t crc64_iso_refl(
00086         uint64_t init_crc,
00087         const unsigned char *buf,
00088         uint64_t len
00089         );
00090
00099 uint64_t crc64_iso_norm(
00100         uint64_t init_crc,
00101         const unsigned char *buf,
00102         uint64_t len
00103         );
00104
00113 uint64_t crc64_jones_refl(
00114         uint64_t init_crc,
00115         const unsigned char *buf,
00116         uint64_t len
00117         );
00118
00127 uint64_t crc64_jones_norm(
00128         uint64_t init_crc,
00129         const unsigned char *buf,
00130         uint64_t len
00131         );
00132
00133 /* Arch specific versions */
00134
00142 uint64_t crc64_ecma_refl_by8(
```

```
00143         uint64_t init_crc,
00144         const unsigned char *buf,
00145         uint64_t len
00146         );
00147
00155 uint64_t crc64_ecma_norm_by8(
00156         uint64_t init_crc,
00157         const unsigned char *buf,
00158         uint64_t len
00159         );
00160
00165 uint64_t crc64_ecma_refl_base(
00166         uint64_t init_crc,
00167         const unsigned char *buf,
00168         uint64_t len
00169         );
00170
00175 uint64_t crc64_ecma_norm_base(
00176         uint64_t init_crc,
00177         const unsigned char *buf,
00178         uint64_t len
00179         );
00180
00188 uint64_t crc64_iso_refl_by8(
00189         uint64_t init_crc,
00190         const unsigned char *buf,
00191         uint64_t len
00192         );
00193
00201 uint64_t crc64_iso_norm_by8(
00202         uint64_t init_crc,
00203         const unsigned char *buf,
00204         uint64_t len
00205         );
00206
00211 uint64_t crc64_iso_refl_base(
00212         uint64_t init_crc,
00213         const unsigned char *buf,
00214         uint64_t len
00215         );
00216
00221 uint64_t crc64_iso_norm_base(
00222         uint64_t init_crc,
00223         const unsigned char *buf,
00224         uint64_t len
00225         );
00226
00234 uint64_t crc64_jones_refl_by8(
00235         uint64_t init_crc,
00236         const unsigned char *buf,
00237         uint64_t len
00238         );
00239
00247 uint64_t crc64_jones_norm_by8(
00248         uint64_t init_crc,
00249         const unsigned char *buf,
00250         uint64_t len
00251         );
00252
00257 uint64_t crc64_jones_refl_base(
00258         uint64_t init_crc,
00259         const unsigned char *buf,
00260         uint64_t len
00261         );
00262
00267 uint64_t crc64_jones_norm_base(
00268         uint64_t init_crc,
00269         const unsigned char *buf,
00270         uint64_t len
00271         );
00272
00273 #ifdef __cplusplus
00274 }
00275 #endif
00276
00277 #endif // _CRC64_H_
```

10.5 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_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_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_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_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.
- 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)

10.5.1 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.

10.5.2 Function Documentation

10.5.2.1 ec_encode_data()

```
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

10.5.2.2 ec_encode_data_base()

```
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.

10.5.2.3 ec_encode_data_update()

```
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

10.5.2.4 ec_encode_data_update_base()

```
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().

10.5.2.5 ec_init_tables()

```
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

10.5.2.6 gf_gen_cauchy1_matrix()

```
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	[m x k] 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

10.5.2.7 gf_gen_rs_matrix()

```
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	[m x k] 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

10.5.2.8 gf_inv()

```
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}

10.5.2.9 gf_invert_matrix()

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

Invert a matrix in GF(2^8)

Attempts to construct an n x n inverse of the input matrix. Returns non-zero if singular. Will always destroy input matrix in process.

Parameters

in	input matrix, destroyed by invert process
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

10.5.2.10 gf_mul()

```
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)

10.5.2.11 gf_vect_dot_prod()

```
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

10.5.2.12 gf_vect_dot_prod_base()

```
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

10.5.2.13 gf_vect_mad()

```
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 >= 64.
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

10.5.2.14 gf_vect_mad_base()

```
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.

10.6 erasure_code.h

Go to the documentation of this file.

```
00001 /**********************************************************************
00002   Copyright(c) 2011-2015 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030
00031 #ifndef _ERASURE_CODE_H_
00032 #define _ERASURE_CODE_H_
00033
00052 #include "gf_vect_mul.h"
00053
00054 #ifdef __cplusplus
00055 extern "C" {
00056 #endif
00057
00074 void ec_init_tables(int k, int rows, unsigned char* a, unsigned char* gftbls);
00075
00098 void ec_encode_data(int len, int k, int rows, unsigned char *gftbls, unsigned char **data,
00099                     unsigned char **coding);
```

```
00100
00106 void ec_encode_data_base(int len, int srcs, int dests, unsigned char *v, unsigned char **src,
00107                          unsigned char **dest);
00108
00131 void ec_encode_data_update(int len, int k, int rows, int vec_i, unsigned char *g_tbls,
00132                            unsigned char *data, unsigned char **coding);
00133
00140 void ec_encode_data_update_base(int len, int k, int rows, int vec_i, unsigned char *v,
00141                                 unsigned char *data, unsigned char **dest);
00142
00164 void gf_vect_dot_prod_base(int len, int vlen, unsigned char *gftbls,
00165                           unsigned char **src, unsigned char *dest);
00166
00187 void gf_vect_dot_prod(int len, int vlen, unsigned char *gftbls,
00188                           unsigned char **src, unsigned char *dest);
00189
00213 void gf_vect_mad(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00214                  unsigned char *dest);
00215
00222 void gf_vect_mad_base(int len, int vec, int vec_i, unsigned char *v, unsigned char *src,
00223                       unsigned char *dest);
00224
00225 // x86 only
00226 #if defined(__i386__) || defined(__x86_64__)
00227
00234 void ec_encode_data_sse(int len, int k, int rows, unsigned char *gftbls, unsigned char **data,
00235                           unsigned char **coding);
00236
00243 void ec_encode_data_avx(int len, int k, int rows, unsigned char *gftbls, unsigned char **data,
00244                           unsigned char **coding);
00245
00252 void ec_encode_data_avx2(int len, int k, int rows, unsigned char *gftbls, unsigned char **data,
00253                           unsigned char **coding);
00254
00262 void ec_encode_data_update_sse(int len, int k, int rows, int vec_i, unsigned char *g_tbls,
00263                                unsigned char *data, unsigned char **coding);
00264
00272 void ec_encode_data_update_avx(int len, int k, int rows, int vec_i, unsigned char *g_tbls,
00273                                unsigned char *data, unsigned char **coding);
00274
00282 void ec_encode_data_update_avx2(int len, int k, int rows, int vec_i, unsigned char *g_tbls,
00283                                 unsigned char *data, unsigned char **coding);
00284
00303 void gf_vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00304                           unsigned char **src, unsigned char *dest);
00305
00324 void gf_vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00325                           unsigned char **src, unsigned char *dest);
00326
00345 void gf_vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00346                           unsigned char **src, unsigned char *dest);
00347
00367 void gf_2vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00368                           unsigned char **src, unsigned char **dest);
00369
00389 void gf_2vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00390                           unsigned char **src, unsigned char **dest);
00391
00411 void gf_2vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00412                           unsigned char **src, unsigned char **dest);
00413
00433 void gf_3vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00434                           unsigned char **src, unsigned char **dest);
00435
00455 void gf_3vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00456                           unsigned char **src, unsigned char **dest);
00457
00477 void gf_3vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00478                           unsigned char **src, unsigned char **dest);
00479
00499 void gf_4vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00500                           unsigned char **src, unsigned char **dest);
00501
00521 void gf_4vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00522                           unsigned char **src, unsigned char **dest);
00523
00543 void gf_4vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00544                           unsigned char **src, unsigned char **dest);
00545
00565 void gf_5vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00566                           unsigned char **src, unsigned char **dest);
```

```
00567
00587 void gf_5vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00588                           unsigned char **src, unsigned char **dest);
00589
00609 void gf_5vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00610                           unsigned char **src, unsigned char **dest);
00611
00631 void gf_6vect_dot_prod_sse(int len, int vlen, unsigned char *gftbls,
00632                           unsigned char **src, unsigned char **dest);
00633
00653 void gf_6vect_dot_prod_avx(int len, int vlen, unsigned char *gftbls,
00654                           unsigned char **src, unsigned char **dest);
00655
00675 void gf_6vect_dot_prod_avx2(int len, int vlen, unsigned char *gftbls,
00676                           unsigned char **src, unsigned char **dest);
00677
00685 void gf_vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00686                      unsigned char *dest);
00694 void gf_vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00695                      unsigned char *dest);
00696
00704 void gf_vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00705                       unsigned char *dest);
00706
00707
00729 void gf_2vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00730                       unsigned char **dest);
00731
00736 void gf_2vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00737                       unsigned char **dest);
00742 void gf_2vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00743                        unsigned char **dest);
00744
00766 void gf_3vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00767                       unsigned char **dest);
00768
00773 void gf_3vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00774                       unsigned char **dest);
00775
00780 void gf_3vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00781                        unsigned char **dest);
00782
00804 void gf_4vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00805                       unsigned char **dest);
00806
00811 void gf_4vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00812                       unsigned char **dest);
00817 void gf_4vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00818                        unsigned char **dest);
00819
00824 void gf_5vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00825                       unsigned char **dest);
00826
00831 void gf_5vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00832                       unsigned char **dest);
00837 void gf_5vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00838                        unsigned char **dest);
00839
00844 void gf_6vect_mad_sse(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00845                       unsigned char **dest);
00850 void gf_6vect_mad_avx(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00851                       unsigned char **dest);
00852
00857 void gf_6vect_mad_avx2(int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src,
00858                        unsigned char **dest);
00859
00860 #endif
00861
00862 /*************************************************************************
00863  * The remaining are lib support functions used in GF(2^8) operations.
00864  */
00865
00874 unsigned char gf_mul(unsigned char a, unsigned char b);
00875
00883 unsigned char gf_inv(unsigned char a);
00884
00909 void gf_gen_rs_matrix(unsigned char *a, int m, int k);
00910
00924 void gf_gen_cauchy1_matrix(unsigned char *a, int m, int k);
00925
00938 int gf_invert_matrix(unsigned char *in, unsigned char *out, const int n);
```

```
00939
00940
00941 /*****************************************************************/
00942
00943 #ifdef __cplusplus
00944 }
00945 #endif
00946
00947 #endif //_ERASURE_CODE_H_
```

10.7 gf_vect_mul.h File Reference

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

Functions

- 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.

10.7.1 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.

10.7.2 Function Documentation

10.7.2.1 gf_vect_mul()

```
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

10.7.2.2 gf_vect_mul_base()

```
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.

10.7.2.3 gf_vect_mul_init()

```
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.

10.8 gf_vect_mul.h

Go to the documentation of this file.

```
00001 /**********************************************************************
00002   Copyright(c) 2011-2015 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030
00031 #ifndef _GF_VECT_MUL_H
00032 #define _GF_VECT_MUL_H
00033
00042 #ifdef __cplusplus
00043 extern "C" {
00044 #endif
00045
00046 // x86 only
00047 #if defined(__i386__) || defined(__x86_64__)
00048
00067 int gf_vect_mul_sse(int len, unsigned char *gftbl, void *src, void *dest);
00068
00069
00088 int gf_vect_mul_avx(int len, unsigned char *gftbl, void *src, void *dest);
00089
00090 #endif
00091
00112 int gf_vect_mul(int len, unsigned char *gftbl, void *src, void *dest);
00113
00114
00125 void gf_vect_mul_init(unsigned char c, unsigned char* gftbl);
00126
00127
00145 void gf_vect_mul_base(int len, unsigned char *a, unsigned char *src,
00146                       unsigned char *dest);
00147
00148 #ifdef __cplusplus
00149 }
00150 #endif
00151
00152 #endif //_GF_VECT_MUL_H
```

10.9 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>
```

Data Structures

- struct isal_huff_histogram

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

  Holds Bit Buffer information.
- struct isal_zlib_header
- struct isal_gzip_header
- 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.
- struct isal_dict

  Structure for holding processed dictionary information.

Enumerations

- enum isal_zstate_state {
  ZSTATE_NEW_HDR , ZSTATE_HDR , ZSTATE_CREATE_HDR , ZSTATE_BODY ,
  ZSTATE_FLUSH_READ_BUFFER , **ZSTATE_FLUSH_ICF_BUFFER** , **ZSTATE_TYPE0_HDR** , 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_FLUSH_ICF_BUFFER** , **ZSTATE_TMP_TYPE0_HDR**
  , ZSTATE_TMP_TYPE0_BODY ,
  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.
- void isal_gzip_header_init (struct isal_gzip_header *gz_hdr)

  Set gzip header default values.
- uint32_t isal_write_gzip_header (struct isal_zstream *stream, struct isal_gzip_header *gz_hdr)

  Write gzip header to output stream.
- uint32_t isal_write_zlib_header (struct isal_zstream *stream, struct isal_zlib_header *z_hdr)

  Write zlib header to output stream.
- 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_process_dict (struct isal_zstream *stream, struct isal_dict *dict_str, uint8_t *dict, uint32_t dict_↩ len)

  Process dictionary to reuse later.
- int isal_deflate_reset_dict (struct isal_zstream *stream, struct isal_dict *dict_str)

  Reset 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_read_gzip_header (struct inflate_state *state, struct isal_gzip_header *gz_hdr)

  Read and return gzip header information.
- int isal_read_zlib_header (struct inflate_state *state, struct isal_zlib_header *zlib_hdr)

  Read and return zlib header information.

- 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.
- uint32_t isal_adler32 (uint32_t init, const unsigned char *buf, uint64_t len)

  Calculate Adler-32 checksum, runs appropriate version.

10.9.1 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 behavior configurable at compile time. These configurable 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 option 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.

10.9.2 Enumeration Type Documentation

10.9.2.1 isal_zstate_state

```
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_TYPE0_BODY	Temporary Type0 block header to be written. Temporary Type0 block body to be written
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.

10.9.3 Function Documentation

10.9.3.1 isal_adler32()

```
uint32_t isal_adler32 (
            uint32_t init,
            const unsigned char * buf,
            uint64_t len )
```

Calculate Adler-32 checksum, runs appropriate version.

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

Parameters

init	initial Adler-32 value
buf	buffer to calculate checksum on
len	buffer length in bytes

Returns

32-bit Adler-32 checksum

10.9.3.2 isal_create_hufftables()

```
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.

10.9.3.3 isal_create_hufftables_subset()

```
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.

10.9.3.4 isal_deflate()

```
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. A full-featured header is supported by the isal_write_{gzip,zlib}_header() functions.

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), ISAL_INVALID_LEVEL_BUF (if the level buffer is not large enough).

10.9.3.5 isal_deflate_init()

```
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

10.9.3.6 isal_deflate_process_dict()

```
int isal_deflate_process_dict (
            struct isal_zstream * stream,
            struct isal_dict * dict_str,
            uint8_t * dict,
            uint32_t dict_len )
```

Process dictionary to reuse later.

Processes a dictionary so that the generated output can be reused to reset a new deflate stream more quickly than isal_deflate_set_dict() alone. This function is paired with isal_deflate_reset_dict() when using the same dictionary on multiple deflate objects. The stream.level must be set prior to calling this function to process the dictionary correctly. 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_str	Structure to hold processed dictionary info to reuse later.
dict	Array containing dictionary to use.
dict_len	Length of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be processed)

10.9.3.7 isal_deflate_reset()

```
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

10.9.3.8 isal_deflate_reset_dict()

```
int isal_deflate_reset_dict (
            struct isal_zstream * stream,
            struct isal_dict * dict_str )
```

Reset compression dictionary to use.

Similar to isal_deflate_set_dict() but on pre-processed dictionary data. Pairing with isal_deflate_process_dict() can reduce the processing time on subsequent compression with dictionary especially on small files.

Like isal_deflate_set_dict(), 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. Changing compression level between dictionary process and reset will cause return of ISAL_INVALID_STATE.

Parameters

stream	Structure holding state information on the compression streams.
dict_str	Structure with pre-processed dictionary info.

Returns

COMP_OK, ISAL_INVALID_STATE or other (dictionary could not be reset)

10.9.3.9 isal_deflate_set_dict()

```
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	Length of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

10.9.3.10 isal_deflate_set_hufftables()

```
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 successful 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.

10.9.3.11 isal_deflate_stateless()

```
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 performance 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), ISAL_INVALID_LEVEL_BUF (if the level buffer is not large enough), STATELESS_OVERFLOW (if output buffer will not fit output).

10.9.3.12 isal_deflate_stateless_init()

```
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

10.9.3.13 isal_gzip_header_init()

```
void isal_gzip_header_init (
            struct isal_gzip_header * gz_hdr )
```

Set gzip header default values.

Parameters

gz_hdr	Gzip header to initialize.

10.9.3.14 isal_inflate()

```
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, invalid data is found, or in the case of zlib formatted data if a dictionary is specified. 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 (checksum may not be updated until decompression is complete). When the crc_flag is set to ISAL_GZIP_NO_HDR_VER or ISAL_ZLIB_NO_HDR_VER, the behavior is the same, except the checksum is verified with the checksum after immediately following the deflate data. If the crc_flag is set to ISAL_GZIP or ISAL_ZLIB, the gzip/zlib header is parsed, state->crc is set to the appropriate checksum, and the checksum is verified. If the crc_flag is set to ISAL_DEFLATE (default), then the data is treated as a raw deflate block.

The element state->hist_bits has values from 0 to 15, where values of 1 to 15 are the log base 2 size of the matching window and 0 is the default with maximum history size.

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_INVALID_BLOCK, ISAL_NEED_DICT, ISAL_INVALID_SYMBOL, ISAL_INVALID_LOOKBACK, ISAL_INVALID_WRAPPER, ISAL_UNSUPPORTED_METHOD, ISAL_INCORRECT_CHECKSUM.

10.9.3.15 isal_inflate_init()

```
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

10.9.3.16 isal_inflate_reset()

```
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

10.9.3.17 isal_inflate_set_dict()

```
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	Length of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

10.9.3.18 isal_inflate_stateless()

```
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. Dictionaries are not supported.

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_NEED_↩ DICT, ISAL_OUT_OVERFLOW (if output buffer ran out of space), ISAL_INVALID_BLOCK, ISAL_INVALID_↩ SYMBOL, ISAL_INVALID_LOOKBACK, ISAL_INVALID_WRAPPER, ISAL_UNSUPPORTED_METHOD, ISAL↩ _INCORRECT_CHECKSUM.

10.9.3.19 isal_read_gzip_header()

```
int isal_read_gzip_header (
            struct inflate_state * state,
            struct isal_gzip_header * gz_hdr )
```

Read and return gzip header information.

On entry state must be initialized and next_in pointing to a gzip compressed buffer. The buffers gz_hdr->extra, gz_↩ hdr->name, gz_hdr->comments and the buffer lengths must be set to record the corresponding field, or set to NULL to disregard that gzip header information. If one of these buffers overflows, the user can reallocate a larger buffer and call this function again to continue reading the header information.

Parameters

state	Structure holding state information on the decompression stream.
gz_hdr	Structure to return data encoded in the gzip header

Returns

ISAL_DECOMP_OK (header was successfully parsed) ISAL_END_INPUT (all input was parsed), ISAL_NAME↩ _OVERFLOW (gz_hdr->name overflowed while parsing), ISAL_COMMENT_OVERFLOW (gz_hdr->comment overflowed while parsing), ISAL_EXTRA_OVERFLOW (gz_hdr->extra overflowed while parsing), ISAL_↩ INVALID_WRAPPER (invalid gzip header found), ISAL_UNSUPPORTED_METHOD (deflate is not the compression method), ISAL_INCORRECT_CHECKSUM (gzip header checksum was incorrect)

10.9.3.20 isal_read_zlib_header()

```
int isal_read_zlib_header (
            struct inflate_state * state,
            struct isal_zlib_header * zlib_hdr )
```

Read and return zlib header information.

On entry state must be initialized and next_in pointing to a zlib compressed buffer.

Parameters

state	Structure holding state information on the decompression stream.
zlib_hdr	Structure to return data encoded in the zlib header

Returns

ISAL_DECOMP_OK (header was successfully parsed), ISAL_END_INPUT (all input was parsed), ISAL_↩ UNSUPPORTED_METHOD (deflate is not the compression method), ISAL_INCORRECT_CHECKSUM (zlib header checksum was incorrect)

10.9.3.21 isal_update_histogram()

```
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.

10.9.3.22 isal_write_gzip_header()

```
uint32_t isal_write_gzip_header (
            struct isal_zstream * stream,
            struct isal_gzip_header * gz_hdr )
```

Write gzip header to output stream.

Writes the gzip header to the output stream. On entry this function assumes that the output buffer has been initialized, so stream->next_out, stream->avail_out and stream->total_out have been set. If the output buffer contains insufficient space, stream is not modified.

Parameters

stream	Structure holding state information on the compression stream.
gz_hdr	Structure holding the gzip header information to encode.

Returns

Returns 0 if the header is successfully written, otherwise returns the minimum size required to successfully write the gzip header to the output buffer.

10.9.3.23 isal_write_zlib_header()

```
uint32_t isal_write_zlib_header (
            struct isal_zstream * stream,
            struct isal_zlib_header * z_hdr )
```

Write zlib header to output stream.

Writes the zlib header to the output stream. On entry this function assumes that the output buffer has been initialized, so stream->next_out, stream->avail_out and stream->total_out have been set. If the output buffer contains insufficient space, stream is not modified.

Parameters

stream	Structure holding state information on the compression stream.
z_hdr	Structure holding the zlib header information to encode.

Returns

Returns 0 if the header is successfully written, otherwise returns the minimum size required to successfully write the zlib header to the output buffer.

10.10 igzip_lib.h

Go to the documentation of this file.

```
00028 *****************************************************************/
00029
00030 #ifndef _IGZIP_H
00031 #define _IGZIP_H
00032
00074 #include <stdint.h>
00075
00076 #ifdef __cplusplus
00077 extern "C" {
00078 #endif
00079
00080 /*******************************************************************************/
00081 /* Deflate Compression Standard Defines */
00082 /*******************************************************************************/
00083 #define IGZIP_K  1024
00084 #define ISAL_DEF_MAX_HDR_SIZE 328
00085 #define ISAL_DEF_MAX_CODE_LEN 15
00086 #define ISAL_DEF_HIST_SIZE (32*IGZIP_K)
00087 #define ISAL_DEF_MAX_HIST_BITS 15
00088 #define ISAL_DEF_MAX_MATCH 258
00089 #define ISAL_DEF_MIN_MATCH 3
00090
00091 #define ISAL_DEF_LIT_SYMBOLS 257
00092 #define ISAL_DEF_LEN_SYMBOLS 29
00093 #define ISAL_DEF_DIST_SYMBOLS 30
00094 #define ISAL_DEF_LIT_LEN_SYMBOLS (ISAL_DEF_LIT_SYMBOLS + ISAL_DEF_LEN_SYMBOLS)
00095
00096 /* Max repeat length, rounded up to 32 byte boundary */
00097 #define ISAL_LOOK_AHEAD ((ISAL_DEF_MAX_MATCH + 31) & ~31)
00098
00099 /*******************************************************************************/
00100 /* Deflate Implementation Specific Defines */
00101 /*******************************************************************************/
00102 /* Note IGZIP_HIST_SIZE must be a power of two */
00103 #ifndef IGZIP_HIST_SIZE
00104 #define IGZIP_HIST_SIZE ISAL_DEF_HIST_SIZE
00105 #endif
00106
00107 #if (IGZIP_HIST_SIZE > ISAL_DEF_HIST_SIZE)
00108 #undef IGZIP_HIST_SIZE
00109 #define IGZIP_HIST_SIZE ISAL_DEF_HIST_SIZE
00110 #endif
00111
00112 #ifdef LONGER_HUFFTABLE
00113 #if (IGZIP_HIST_SIZE > 8 * IGZIP_K)
00114 #undef IGZIP_HIST_SIZE
00115 #define IGZIP_HIST_SIZE (8 * IGZIP_K)
00116 #endif
00117 #endif
00118
00119 #define ISAL_LIMIT_HASH_UPDATE
00120
00121 #define IGZIP_HASH8K_HASH_SIZE (8 * IGZIP_K)
00122 #define IGZIP_HASH_HIST_SIZE IGZIP_HIST_SIZE
00123 #define IGZIP_HASH_MAP_HASH_SIZE IGZIP_HIST_SIZE
00124
00125 #define IGZIP_LVL0_HASH_SIZE  (8 * IGZIP_K)
00126 #define IGZIP_LVL1_HASH_SIZE  IGZIP_HASH8K_HASH_SIZE
00127 #define IGZIP_LVL2_HASH_SIZE  IGZIP_HASH_HIST_SIZE
00128 #define IGZIP_LVL3_HASH_SIZE  IGZIP_HASH_MAP_HASH_SIZE
00129
00130 #ifdef LONGER_HUFFTABLE
00131 enum {IGZIP_DIST_TABLE_SIZE = 8*1024};
00132
00133 /* DECODE_OFFSET is dist code index corresponding to DIST_TABLE_SIZE + 1 */
00134 enum { IGZIP_DECODE_OFFSET = 26 };
00135 #else
00136 enum {IGZIP_DIST_TABLE_SIZE = 2};
00137 /* DECODE_OFFSET is dist code index corresponding to DIST_TABLE_SIZE + 1 */
00138 enum { IGZIP_DECODE_OFFSET = 0 };
00139 #endif
00140 enum {IGZIP_LEN_TABLE_SIZE = 256};
00141 enum {IGZIP_LIT_TABLE_SIZE = ISAL_DEF_LIT_SYMBOLS};
00142
00143 #define IGZIP_HUFFTABLE_CUSTOM 0
00144 #define IGZIP_HUFFTABLE_DEFAULT 1
00145 #define IGZIP_HUFFTABLE_STATIC 2
00146
00147 /* Flush Flags */
00148 #define NO_FLUSH        0       /* Default */
00149 #define SYNC_FLUSH      1
```

```
00150 #define FULL_FLUSH      2
00151 #define FINISH_FLUSH    0       /* Deprecated */
00152
00153 /* Gzip Flags */
00154 #define IGZIP_DEFLATE   0       /* Default */
00155 #define IGZIP_GZIP      1
00156 #define IGZIP_GZIP_NO_HDR       2
00157 #define IGZIP_ZLIB      3
00158 #define IGZIP_ZLIB_NO_HDR       4
00159
00160 /* Compression Return values */
00161 #define COMP_OK 0
00162 #define INVALID_FLUSH -7
00163 #define INVALID_PARAM -8
00164 #define STATELESS_OVERFLOW -1
00165 #define ISAL_INVALID_OPERATION -9
00166 #define ISAL_INVALID_STATE -3
00167 #define ISAL_INVALID_LEVEL -4   /* Invalid Compression level set */
00168 #define ISAL_INVALID_LEVEL_BUF -5 /* Invalid buffer specified for the compression level */
00169
00176 /* When the state is set to ZSTATE_NEW_HDR or TMP_ZSTATE_NEW_HEADER, the
00177  * hufftable being used for compression may be swapped
00178  */
00179 enum isal_zstate_state {
00180         ZSTATE_NEW_HDR,
00181         ZSTATE_HDR,
00182         ZSTATE_CREATE_HDR,
00183         ZSTATE_BODY,
00184         ZSTATE_FLUSH_READ_BUFFER,
00185         ZSTATE_FLUSH_ICF_BUFFER,
00186         ZSTATE_TYPE0_HDR,
00187         ZSTATE_TYPE0_BODY,
00188         ZSTATE_SYNC_FLUSH,
00189         ZSTATE_FLUSH_WRITE_BUFFER,
00190         ZSTATE_TRL,
00191         ZSTATE_END,
00192         ZSTATE_TMP_NEW_HDR,
00193         ZSTATE_TMP_HDR,
00194         ZSTATE_TMP_CREATE_HDR,
00195         ZSTATE_TMP_BODY,
00196         ZSTATE_TMP_FLUSH_READ_BUFFER,
00197         ZSTATE_TMP_FLUSH_ICF_BUFFER,
00198         ZSTATE_TMP_TYPE0_HDR,
00199         ZSTATE_TMP_TYPE0_BODY,
00200         ZSTATE_TMP_SYNC_FLUSH,
00201         ZSTATE_TMP_FLUSH_WRITE_BUFFER,
00202         ZSTATE_TMP_TRL,
00203         ZSTATE_TMP_END
00204 };
00205
00206 /* Offset used to switch between TMP states and non-tmp states */
00207 #define ZSTATE_TMP_OFFSET ZSTATE_TMP_HDR - ZSTATE_HDR
00208
00209 /*****************************************************************************/
00210 /* Inflate Implementation Specific Defines */
00211 /*****************************************************************************/
00212 #define ISAL_DECODE_LONG_BITS 12
00213 #define ISAL_DECODE_SHORT_BITS 10
00214
00215 /* Current state of decompression */
00216 enum isal_block_state {
00217         ISAL_BLOCK_NEW_HDR,     /* Just starting a new block */
00218         ISAL_BLOCK_HDR,         /* In the middle of reading in a block header */
00219         ISAL_BLOCK_TYPE0,       /* Decoding a type 0 block */
00220         ISAL_BLOCK_CODED,       /* Decoding a huffman coded block */
00221         ISAL_BLOCK_INPUT_DONE,  /* Decompression of input is completed */
00222         ISAL_BLOCK_FINISH,      /* Decompression of input is completed and all data has been flushed to
      output */
00223         ISAL_GZIP_EXTRA_LEN,
00224         ISAL_GZIP_EXTRA,
00225         ISAL_GZIP_NAME,
00226         ISAL_GZIP_COMMENT,
00227         ISAL_GZIP_HCRC,
00228         ISAL_ZLIB_DICT,
00229         ISAL_CHECKSUM_CHECK,
00230 };
00231
00232
00233 /* Inflate Flags */
00234 #define ISAL_DEFLATE    0       /* Default */
00235 #define ISAL_GZIP       1
```

```
00236 #define ISAL_GZIP_NO_HDR        2
00237 #define ISAL_ZLIB       3
00238 #define ISAL_ZLIB_NO_HDR        4
00239 #define ISAL_ZLIB_NO_HDR_VER    5
00240 #define ISAL_GZIP_NO_HDR_VER    6
00241
00242 /* Inflate Return values */
00243 #define ISAL_DECOMP_OK 0        /* No errors encountered while decompressing */
00244 #define ISAL_END_INPUT 1        /* End of input reached */
00245 #define ISAL_OUT_OVERFLOW 2     /* End of output reached */
00246 #define ISAL_NAME_OVERFLOW 3    /* End of gzip name buffer reached */
00247 #define ISAL_COMMENT_OVERFLOW 4 /* End of gzip name buffer reached */
00248 #define ISAL_EXTRA_OVERFLOW 5   /* End of extra buffer reached */
00249 #define ISAL_NEED_DICT 6 /* Stream needs a dictionary to continue */
00250 #define ISAL_INVALID_BLOCK -1   /* Invalid deflate block found */
00251 #define ISAL_INVALID_SYMBOL -2  /* Invalid deflate symbol found */
00252 #define ISAL_INVALID_LOOKBACK -3        /* Invalid lookback distance found */
00253 #define ISAL_INVALID_WRAPPER -4 /* Invalid gzip/zlib wrapper found */
00254 #define ISAL_UNSUPPORTED_METHOD -5      /* Gzip/zlib wrapper specifies unsupported compress method */
00255 #define ISAL_INCORRECT_CHECKSUM -6 /* Incorrect checksum found */
00256
00257 /*******************************************************************************/
00258 /* Compression structures */
00259 /*******************************************************************************/
00261 struct isal_huff_histogram {
00262         uint64_t lit_len_histogram[ISAL_DEF_LIT_LEN_SYMBOLS];
00263         uint64_t dist_histogram[ISAL_DEF_DIST_SYMBOLS];
00264         uint16_t hash_table[IGZIP_LVL0_HASH_SIZE];
00265 };
00266
00267 struct isal_mod_hist {
00268     uint32_t d_hist[30];
00269     uint32_t ll_hist[513];
00270 };
00271
00272 #define ISAL_DEF_MIN_LEVEL 0
00273 #define ISAL_DEF_MAX_LEVEL 3
00274
00275 /* Defines used set level data sizes */
00276 /* has to be at least sizeof(struct level_buf) + sizeof(struct lvlX_buf */
00277 #define ISAL_DEF_LVL0_REQ 0
00278 #define ISAL_DEF_LVL1_REQ (4 * IGZIP_K + 2 * IGZIP_LVL1_HASH_SIZE)
00279 #define ISAL_DEF_LVL1_TOKEN_SIZE 4
00280 #define ISAL_DEF_LVL2_REQ (4 * IGZIP_K + 2 * IGZIP_LVL2_HASH_SIZE)
00281 #define ISAL_DEF_LVL2_TOKEN_SIZE 4
00282 #define ISAL_DEF_LVL3_REQ 4 * IGZIP_K + 4 * 4 * IGZIP_K + 2 * IGZIP_LVL3_HASH_SIZE
00283 #define ISAL_DEF_LVL3_TOKEN_SIZE 4
00284
00285 /* Data sizes for level specific data options */
00286 #define ISAL_DEF_LVL0_MIN ISAL_DEF_LVL0_REQ
00287 #define ISAL_DEF_LVL0_SMALL ISAL_DEF_LVL0_REQ
00288 #define ISAL_DEF_LVL0_MEDIUM ISAL_DEF_LVL0_REQ
00289 #define ISAL_DEF_LVL0_LARGE ISAL_DEF_LVL0_REQ
00290 #define ISAL_DEF_LVL0_EXTRA_LARGE ISAL_DEF_LVL0_REQ
00291 #define ISAL_DEF_LVL0_DEFAULT ISAL_DEF_LVL0_REQ
00292
00293 #define ISAL_DEF_LVL1_MIN (ISAL_DEF_LVL1_REQ + ISAL_DEF_LVL1_TOKEN_SIZE * 1 * IGZIP_K)
00294 #define ISAL_DEF_LVL1_SMALL (ISAL_DEF_LVL1_REQ + ISAL_DEF_LVL1_TOKEN_SIZE * 16 * IGZIP_K)
00295 #define ISAL_DEF_LVL1_MEDIUM (ISAL_DEF_LVL1_REQ + ISAL_DEF_LVL1_TOKEN_SIZE * 32 * IGZIP_K)
00296 #define ISAL_DEF_LVL1_LARGE (ISAL_DEF_LVL1_REQ + ISAL_DEF_LVL1_TOKEN_SIZE * 64 * IGZIP_K)
00297 #define ISAL_DEF_LVL1_EXTRA_LARGE (ISAL_DEF_LVL1_REQ + ISAL_DEF_LVL1_TOKEN_SIZE * 128 * IGZIP_K)
00298 #define ISAL_DEF_LVL1_DEFAULT ISAL_DEF_LVL1_LARGE
00299
00300 #define ISAL_DEF_LVL2_MIN (ISAL_DEF_LVL2_REQ + ISAL_DEF_LVL2_TOKEN_SIZE * 1 * IGZIP_K)
00301 #define ISAL_DEF_LVL2_SMALL (ISAL_DEF_LVL2_REQ + ISAL_DEF_LVL2_TOKEN_SIZE * 16 * IGZIP_K)
00302 #define ISAL_DEF_LVL2_MEDIUM (ISAL_DEF_LVL2_REQ + ISAL_DEF_LVL2_TOKEN_SIZE * 32 * IGZIP_K)
00303 #define ISAL_DEF_LVL2_LARGE (ISAL_DEF_LVL2_REQ + ISAL_DEF_LVL2_TOKEN_SIZE * 64 * IGZIP_K)
00304 #define ISAL_DEF_LVL2_EXTRA_LARGE (ISAL_DEF_LVL2_REQ + ISAL_DEF_LVL2_TOKEN_SIZE * 128 * IGZIP_K)
00305 #define ISAL_DEF_LVL2_DEFAULT ISAL_DEF_LVL2_LARGE
00306
00307 #define ISAL_DEF_LVL3_MIN (ISAL_DEF_LVL3_REQ + ISAL_DEF_LVL3_TOKEN_SIZE * 1 * IGZIP_K)
00308 #define ISAL_DEF_LVL3_SMALL (ISAL_DEF_LVL3_REQ + ISAL_DEF_LVL3_TOKEN_SIZE * 16 * IGZIP_K)
00309 #define ISAL_DEF_LVL3_MEDIUM (ISAL_DEF_LVL3_REQ + ISAL_DEF_LVL3_TOKEN_SIZE * 32 * IGZIP_K)
00310 #define ISAL_DEF_LVL3_LARGE (ISAL_DEF_LVL3_REQ + ISAL_DEF_LVL3_TOKEN_SIZE * 64 * IGZIP_K)
00311 #define ISAL_DEF_LVL3_EXTRA_LARGE (ISAL_DEF_LVL3_REQ + ISAL_DEF_LVL3_TOKEN_SIZE * 128 * IGZIP_K)
00312 #define ISAL_DEF_LVL3_DEFAULT ISAL_DEF_LVL3_LARGE
00313
00314 #define IGZIP_NO_HIST 0
00315 #define IGZIP_HIST 1
00316 #define IGZIP_DICT_HIST 2
00317 #define IGZIP_DICT_HASH_SET 3
```

```
00318
00320 struct BitBuf2 {
00321         uint64_t m_bits;
00322         uint32_t m_bit_count;
00323         uint8_t *m_out_buf;
00324         uint8_t *m_out_end;
00325         uint8_t *m_out_start;
00326 };
00327
00328 struct isal_zlib_header {
00329         uint32_t info;
00330         uint32_t level;
00331         uint32_t dict_id;
00332         uint32_t dict_flag;
00333 };
00334
00335 struct isal_gzip_header {
00336         uint32_t text;
00337         uint32_t time;
00338         uint32_t xflags;
00339         uint32_t os;
00340         uint8_t *extra;
00341         uint32_t extra_buf_len;
00342         uint32_t extra_len;
00343         char *name;
00344         uint32_t name_buf_len;
00345         char *comment;
00346         uint32_t comment_buf_len;
00347         uint32_t hcrc;
00348         uint32_t flags;
00349 };
00350
00351 /* Variable prefixes:
00352  * b_ : Measured wrt the start of the buffer
00353  * f_ : Measured wrt the start of the file (aka file_start)
00354  */
00355
00357 struct isal_zstate {
00358         uint32_t total_in_start;
00359         uint32_t block_next;
00360         uint32_t block_end;
00361         uint32_t dist_mask;
00362         uint32_t hash_mask;
00363         enum isal_zstate_state state;
00364         struct BitBuf2 bitbuf;
00365         uint32_t crc;
00366         uint8_t has_wrap_hdr;
00367         uint8_t has_eob_hdr;
00368         uint8_t has_eob;
00369         uint8_t has_hist;
00370         uint16_t has_level_buf_init;
00371         uint32_t count;
00372         uint8_t tmp_out_buff[16];
00373         uint32_t tmp_out_start;
00374         uint32_t tmp_out_end;
00375         uint32_t b_bytes_valid;
00376         uint32_t b_bytes_processed;
00377         uint8_t buffer[2 * IGZIP_HIST_SIZE + ISAL_LOOK_AHEAD];
00378
00379         /* Stream should be setup such that the head is cache aligned*/
00380         uint16_t head[IGZIP_LVL0_HASH_SIZE];
00381 };
00382
00384 struct isal_hufftables {
00385
00386         uint8_t deflate_hdr[ISAL_DEF_MAX_HDR_SIZE];
00387         uint32_t deflate_hdr_count;
00388         uint32_t deflate_hdr_extra_bits;
00389         uint32_t dist_table[IGZIP_DIST_TABLE_SIZE];
00390         uint32_t len_table[IGZIP_LEN_TABLE_SIZE];
00391         uint16_t lit_table[IGZIP_LIT_TABLE_SIZE];
00392         uint8_t lit_table_sizes[IGZIP_LIT_TABLE_SIZE];
00393         uint16_t dcodes[30 - IGZIP_DECODE_OFFSET];
00394         uint8_t dcodes_sizes[30 - IGZIP_DECODE_OFFSET];
00395
00396 };
00397
00399 struct isal_zstream {
00400         uint8_t *next_in;
00401         uint32_t avail_in;
00402         uint32_t total_in;
```

```
00403
00404         uint8_t *next_out;
00405         uint32_t avail_out;
00406         uint32_t total_out;
00407
00408         struct isal_hufftables *hufftables;
00409         uint32_t level;
00410         uint32_t level_buf_size;
00411         uint8_t * level_buf;
00412         uint16_t end_of_stream;
00413         uint16_t flush;
00414         uint16_t gzip_flag;
00415         uint16_t hist_bits;
00416         struct isal_zstate internal_state;
00417 };
00418
00419 /*****************************************************************************/
00420 /* Inflate structures */
00421 /*****************************************************************************/
00422 /*
00423  * Inflate_huff_code data structures are used to store a Huffman code for fast
00424  * lookup. It works by performing a lookup in small_code_lookup that hopefully
00425  * yields the correct symbol. Otherwise a lookup into long_code_lookup is
00426  * performed to find the correct symbol. The details of how this works follows:
00427  *
00428  * Let i be some index into small_code_lookup and let e be the associated
00429  * element.  Bit 15 in e is a flag. If bit 15 is not set, then index i contains
00430  * a Huffman code for a symbol which has length at most DECODE_LOOKUP_SIZE. Bits
00431  * 0 through 8 are the symbol associated with that code and bits 9 through 12 of
00432  * e represent the number of bits in the code. If bit 15 is set, the i
00433  * corresponds to the first DECODE_LOOKUP_SIZE bits of a Huffman code which has
00434  * length longer than DECODE_LOOKUP_SIZE. In this case, bits 0 through 8
00435  * represent an offset into long_code_lookup table and bits 9 through 12
00436  * represent the maximum length of a Huffman code starting with the bits in the
00437  * index i. The offset into long_code_lookup is for an array associated with all
00438  * codes which start with the bits in i.
00439  *
00440  * The elements of long_code_lookup are in the same format as small_code_lookup,
00441  * except bit 15 is never set. Let i be a number made up of DECODE_LOOKUP_SIZE
00442  * bits.  Then all Huffman codes which start with DECODE_LOOKUP_SIZE bits are
00443  * stored in an array starting at index h in long_code_lookup. This index h is
00444  * stored in bits 0 through 9 at index i in small_code_lookup. The index j is an
00445  * index of this array if the number of bits contained in j and i is the number
00446  * of bits in the longest huff_code starting with the bits of i. The symbol
00447  * stored at index j is the symbol whose huffcode can be found in (j «
00448  * DECODE_LOOKUP_SIZE) | i. Note these arrays will be stored sorted in order of
00449  * maximum Huffman code length.
00450  *
00451  * The following are explanations for sizes of the tables:
00452  *
00453  * Since small_code_lookup is a lookup on DECODE_LOOKUP_SIZE bits, it must have
00454  * size 2^DECODE_LOOKUP_SIZE.
00455  *
00456  * To determine the amount of memory required for long_code_lookup, note that
00457  * any element of long_code_lookup corresponds to a code, a duplicate of an
00458  * existing code, or a invalid code. Since deflate Huffman are stored such that
00459  * the code size and the code value form an increasing function, the number of
00460  * duplicates is maximized when all the duplicates are contained in a single
00461  * array, thus there are at most 2^(15 - DECODE_LOOKUP_SIZE) -
00462  * (DECODE_LOOKUP_SIZE + 1) duplicate elements. Similarly the number of invalid
00463  * elements is maximized at 2^(15 - DECODE_LOOKUP_SIZE) - 2^(floor((15 -
00464  * DECODE_LOOKUP_SIZE)/2) - 2^(ceil((15 - DECODE_LOOKUP_SIZE)/2) + 1. Thus the
00465  * amount of memory required is: NUM_CODES + 2^(16 - DECODE_LOOKUP_SIZE) -
00466  * (DECODE_LOOKUP_SIZE + 1) - 2^(floor((15 - DECODE_LOOKUP_SIZE)/2) -
00467  * 2^(ceil((15 - DECODE_LOOKUP_SIZE)/2) + 1. The values used below are those
00468  * values rounded up to the nearest 16 byte boundary
00469  *
00470  * Note that DECODE_LOOKUP_SIZE can be any length even though the offset in
00471  * small_lookup_code is 9 bits long because the increasing relationship between
00472  * code length and code value forces the maximum offset to be less than 288.
00473  */
00474
00475 /* In the following defines, L stands for LARGE and S for SMALL */
00476 #define ISAL_L_REM (21 - ISAL_DECODE_LONG_BITS)
00477 #define ISAL_S_REM (15 - ISAL_DECODE_SHORT_BITS)
00478
00479 #define ISAL_L_DUP ((1 « ISAL_L_REM) - (ISAL_L_REM + 1))
00480 #define ISAL_S_DUP ((1 « ISAL_S_REM) - (ISAL_S_REM + 1))
00481
00482 #define ISAL_L_UNUSED ((1 « ISAL_L_REM) - (1 « ((ISAL_L_REM)/2)) - (1 « ((ISAL_L_REM + 1)/2)) + 1)
00483 #define ISAL_S_UNUSED ((1 « ISAL_S_REM) - (1 « ((ISAL_S_REM)/2)) - (1 « ((ISAL_S_REM + 1)/2)) + 1)
```

```
00484
00485 #define ISAL_L_SIZE (ISAL_DEF_LIT_LEN_SYMBOLS + ISAL_L_DUP + ISAL_L_UNUSED)
00486 #define ISAL_S_SIZE (ISAL_DEF_DIST_SYMBOLS + ISAL_S_DUP + ISAL_S_UNUSED)
00487
00488 #define ISAL_HUFF_CODE_LARGE_LONG_ALIGNED (ISAL_L_SIZE + (-ISAL_L_SIZE & 0xf))
00489 #define ISAL_HUFF_CODE_SMALL_LONG_ALIGNED (ISAL_S_SIZE + (-ISAL_S_SIZE & 0xf))
00490
00491 /* Large lookup table for decoding huffman codes */
00492 struct inflate_huff_code_large {
00493         uint32_t short_code_lookup[1 << (ISAL_DECODE_LONG_BITS)];
00494         uint16_t long_code_lookup[ISAL_HUFF_CODE_LARGE_LONG_ALIGNED];
00495 };
00496
00497 /* Small lookup table for decoding huffman codes */
00498 struct inflate_huff_code_small {
00499         uint16_t short_code_lookup[1 << (ISAL_DECODE_SHORT_BITS)];
00500         uint16_t long_code_lookup[ISAL_HUFF_CODE_SMALL_LONG_ALIGNED];
00501 };
00502
00504 struct inflate_state {
00505         uint8_t *next_out;
00506         uint32_t avail_out;
00507         uint32_t total_out;
00508         uint8_t *next_in;
00509         uint64_t read_in;
00510         uint32_t avail_in;
00511         int32_t read_in_length;
00512         struct inflate_huff_code_large lit_huff_code;
00513         struct inflate_huff_code_small dist_huff_code;
00514         enum isal_block_state block_state;
00515         uint32_t dict_length;
00516         uint32_t bfinal;
00517         uint32_t crc_flag;
00518         uint32_t crc;
00519         uint32_t hist_bits;
00520         union {
00521                 int32_t type0_block_len;
00522                 int32_t count;
00523                 uint32_t dict_id;
00524         };
00525         int32_t write_overflow_lits;
00526         int32_t write_overflow_len;
00527         int32_t copy_overflow_length;
00528         int32_t copy_overflow_distance;
00529         int16_t wrapper_flag;
00530         int16_t tmp_in_size;
00531         int32_t tmp_out_valid;
00532         int32_t tmp_out_processed;
00533         uint8_t tmp_in_buffer[ISAL_DEF_MAX_HDR_SIZE];
00534         uint8_t tmp_out_buffer[2 * ISAL_DEF_HIST_SIZE + ISAL_LOOK_AHEAD];
00535 };
00536
00537 /*****************************************************************************/
00538 /* Compression functions */
00539 /*****************************************************************************/
00551 void isal_update_histogram(uint8_t * in_stream, int length, struct isal_huff_histogram * histogram);
00552
00553
00564 int isal_create_hufftables(struct isal_hufftables * hufftables,
00565                            struct isal_huff_histogram * histogram);
00566
00577 int isal_create_hufftables_subset(struct isal_hufftables * hufftables,
00578                                   struct isal_huff_histogram * histogram);
00579
00586 void isal_deflate_init(struct isal_zstream *stream);
00587
00597 void isal_deflate_reset(struct isal_zstream *stream);
00598
00599
00605 void isal_gzip_header_init(struct isal_gzip_header *gz_hdr);
00606
00622 uint32_t isal_write_gzip_header(struct isal_zstream * stream, struct isal_gzip_header *gz_hdr);
00623
00639 uint32_t isal_write_zlib_header(struct isal_zstream * stream, struct isal_zlib_header *z_hdr);
00640
00661 int isal_deflate_set_hufftables(struct isal_zstream *stream,
00662                                 struct isal_hufftables *hufftables, int type);
00663
00670 void isal_deflate_stateless_init(struct isal_zstream *stream);
00671
00672
```

```
00687 int isal_deflate_set_dict(struct isal_zstream *stream, uint8_t *dict, uint32_t dict_len);
00688
00691 struct isal_dict {
00692         uint32_t params;
00693         uint32_t level;
00694         uint32_t hist_size;
00695         uint32_t hash_size;
00696         uint8_t history[ISAL_DEF_HIST_SIZE];
00697         uint16_t hashtable[IGZIP_LVL3_HASH_SIZE];
00698 };
00699
00718 int isal_deflate_process_dict(struct isal_zstream *stream, struct isal_dict *dict_str,
00719                         uint8_t *dict, uint32_t dict_len);
00720
00738 int isal_deflate_reset_dict(struct isal_zstream *stream, struct isal_dict *dict_str);
00739
00740
00795 int isal_deflate(struct isal_zstream *stream);
00796
00797
00825 int isal_deflate_stateless(struct isal_zstream *stream);
00826
00827
00828 /*****************************************************************************/
00829 /* Inflate functions */
00830 /*****************************************************************************/
00837 void isal_inflate_init(struct inflate_state *state);
00838
00845 void isal_inflate_reset(struct inflate_state *state);
00846
00860 int isal_inflate_set_dict(struct inflate_state *state, uint8_t *dict, uint32_t dict_len);
00861
00883 int isal_read_gzip_header (struct inflate_state *state, struct isal_gzip_header *gz_hdr);
00884
00898 int isal_read_zlib_header (struct inflate_state *state, struct isal_zlib_header *zlib_hdr);
00899
00946 int isal_inflate(struct inflate_state *state);
00947
00968 int isal_inflate_stateless(struct inflate_state *state);
00969
00970 /*****************************************************************************/
00971 /* Other functions */
00972 /*****************************************************************************/
00985 uint32_t isal_adler32(uint32_t init, const unsigned char *buf, uint64_t len);
00986
00987 #ifdef __cplusplus
00988 }
00989 #endif
00990 #endif  /* ifndef _IGZIP_H */
```

10.11 mem_routines.h File Reference

Interface to storage mem operations.

```
#include <stddef.h>
```

Functions

- int isal_zero_detect (void *mem, size_t len)

  Detect if a memory region is all zero.

10.11.1 Detailed Description

Interface to storage mem operations.

Defines the interface for vector versions of common memory functions.

10.11.2 Function Documentation

10.11.2.1 isal_zero_detect()

```
int isal_zero_detect (
             void * mem,
             size_t len )
```

Detect if a memory region is all zero.

Zero detect function with optimizations for large blocks > 128 bytes

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

10.12 mem_routines.h

Go to the documentation of this file.

```
00001 /**********************************************************************
00002   Copyright(c) 2011-2018 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030 #include <stddef.h>
00031
00040 #ifndef _MEM_ROUTINES_H_
00041 #define _MEM_ROUTINES_H_
00042
00043 #ifdef __cplusplus
00044 extern "C" {
00045 #endif
```

```
00046
00057 int isal_zero_detect(void *mem, size_t len);
00058
00059 #ifdef __cplusplus
00060 }
00061 #endif
00062
00063 #endif // _MEM_ROUTINES_H_
00064
```

10.13 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 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.

10.13.1 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.

10.13.2 Function Documentation

10.13.2.1 pq_check()

```
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

10.13.2.2 pq_check_base()

```
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

10.13.2.3 pq_gen()

```
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

10.13.2.4 pq_gen_base()

```
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

10.13.2.5 xor_check()

```
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

10.13.2.6 xor_check_base()

```
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

10.13.2.7 xor_gen()

```
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

10.13.2.8 xor_gen_base()

```
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

10.14 raid.h

Go to the documentation of this file.

```
00001 /**********************************************************************
00002   Copyright(c) 2011-2015 Intel Corporation All rights reserved.
00003
00004   Redistribution and use in source and binary forms, with or without
00005   modification, are permitted provided that the following conditions
00006   are met:
00007     * Redistributions of source code must retain the above copyright
00008       notice, this list of conditions and the following disclaimer.
00009     * Redistributions in binary form must reproduce the above copyright
00010       notice, this list of conditions and the following disclaimer in
00011       the documentation and/or other materials provided with the
00012       distribution.
00013     * Neither the name of Intel Corporation nor the names of its
00014       contributors may be used to endorse or promote products derived
00015       from this software without specific prior written permission.
00016
00017   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00018   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00019   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00020   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00021   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00023   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00024   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00025   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00026   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00027   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00028 **********************************************************************/
00029
00030
00031 #ifndef _RAID_H_
00032 #define _RAID_H_
00033
00043 #ifdef __cplusplus
00044 extern "C" {
00045 #endif
00046
00047 /* Multi-binary functions */
00048
00064 int xor_gen(int vects, int len, void **array);
00065
00066
00081 int xor_check(int vects, int len, void **array);
00082
00083
00101 int pq_gen(int vects, int len, void **array);
```

```
00102
00103
00119 int pq_check(int vects, int len, void **array);
00120
00121
00122 /* Arch specific versions */
00123 // x86 only
00124 #if defined(__i386__) || defined(__x86_64__)
00125
00139 int xor_gen_sse(int vects, int len, void **array);
00140
00141
00155 int xor_gen_avx(int vects, int len, void **array);
00156
00157
00170 int xor_check_sse(int vects, int len, void **array);
00171
00172
00188 int pq_gen_sse(int vects, int len, void **array);
00189
00190
00206 int pq_gen_avx(int vects, int len, void **array);
00207
00208
00224 int pq_gen_avx2(int vects, int len, void **array);
00225
00226
00239 int pq_check_sse(int vects, int len, void **array);
00240
00241 #endif
00242
00256 int pq_gen_base(int vects, int len, void **array);
00257
00258
00270 int xor_gen_base(int vects, int len, void **array);
00271
00272
00284 int xor_check_base(int vects, int len, void **array);
00285
00286
00299 int pq_check_base(int vects, int len, void **array);
00300
00301 #ifdef __cplusplus
00302 }
00303 #endif
00304
00305 #endif //_RAID_H_
```

10.15 isa-l.h File Reference

Include for ISA-L library.

```
#include <isa-l/crc.h>
#include <isa-l/crc64.h>
#include <isa-l/erasure_code.h>
#include <isa-l/gf_vect_mul.h>
#include <isa-l/igzip_lib.h>
#include <isa-l/mem_routines.h>
#include <isa-l/raid.h>
```

10.15.1 Detailed Description

Include for ISA-L library.

10.16 isa-l.h

Go to the documentation of this file.

```
00001
00007 #ifndef _ISAL_H_
00008 #define _ISAL_H_
00009
00010 #define ISAL_MAJOR_VERSION 2
00011 #define ISAL_MINOR_VERSION 30
00012 #define ISAL_PATCH_VERSION 0
00013 #define ISAL_MAKE_VERSION(maj, min, patch)  ((maj) * 0x10000 + (min) * 0x100 + (patch))
00014 #define ISAL_VERSION ISAL_MAKE_VERSION(ISAL_MAJOR_VERSION, ISAL_MINOR_VERSION, ISAL_PATCH_VERSION)
00015
00016 #include <isa-l/crc.h>
00017 #include <isa-l/crc64.h>
00018 #include <isa-l/erasure_code.h>
00019 #include <isa-l/gf_vect_mul.h>
00020 #include <isa-l/igzip_lib.h>
00021 #include <isa-l/mem_routines.h>
00022 #include <isa-l/raid.h>
00023 #endif //_ISAL_H_
```

Index