



# Intel® Intelligent Storage Acceleration Library (Intel® ISA-L)

API Reference Manual - Version 2.12

---

*October 17, 2014*

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S LICENSE AGREEMENT FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2011 - 2014 Intel Corporation. All rights reserved.

---

# Contents

<b>1</b>	<b>Storage Library</b>	<b>1</b>
1.1	About This Document . . . . .	1
1.2	Overview . . . . .	1
1.3	RAID Functions . . . . .	1
1.4	Erasure Code Functions . . . . .	2
1.5	CRC Functions . . . . .	2
1.6	Multi-buffer Hashing Functions . . . . .	2
1.7	Alignment for Input Parameters . . . . .	3
1.8	System Requirements . . . . .	3
<b>2</b>	<b>Function Version Numbers</b>	<b>5</b>
2.1	Function Version Numbers . . . . .	5
2.2	Function Version Numbers Tables . . . . .	6
<b>3</b>	<b>Instruction Set Requirements</b>	<b>11</b>
<b>4</b>	<b>Data Structure Index</b>	<b>21</b>
4.1	Data Structures . . . . .	21
<b>5</b>	<b>File Index</b>	<b>24</b>
5.1	File List . . . . .	24
<b>6</b>	<b>Data Structure Documentation</b>	<b>26</b>
6.1	BitBuf2 Struct Reference . . . . .	26
6.1.1	Detailed Description . . . . .	26
6.2	JOB_MD5 Struct Reference . . . . .	26
6.2.1	Detailed Description . . . . .	27
6.3	JOB_SHA1 Struct Reference . . . . .	27
6.3.1	Detailed Description . . . . .	28
6.4	JOB_SHA256 Struct Reference . . . . .	28
6.4.1	Detailed Description . . . . .	29
6.5	JOB_SHA512 Struct Reference . . . . .	29
6.5.1	Detailed Description . . . . .	29
6.6	LZ_State1 Struct Reference . . . . .	30
6.6.1	Detailed Description . . . . .	31
6.7	LZ_Stream1 Struct Reference . . . . .	31

---

6.7.1	Detailed Description	32
6.8	MD5_ARGS_X8 Struct Reference	32
6.8.1	Detailed Description	32
6.9	MD5_ARGS_X8X2 Struct Reference	33
6.9.1	Detailed Description	33
6.10	MD5_HASH_CTX Struct Reference	33
6.10.1	Detailed Description	34
6.11	MD5_HASH_CTX_MGR Struct Reference	34
6.11.1	Detailed Description	34
6.12	MD5_HMAC_LANE_DATA Struct Reference	34
6.12.1	Detailed Description	35
6.13	MD5_JOB Struct Reference	35
6.13.1	Detailed Description	35
6.14	MD5_LANE_DATA Struct Reference	35
6.14.1	Detailed Description	36
6.15	MD5_MB_ARGS_X16 Struct Reference	36
6.15.1	Detailed Description	36
6.16	MD5_MB_JOB_MGR Struct Reference	36
6.16.1	Detailed Description	36
6.17	MD5_MB_MGR Struct Reference	37
6.17.1	Detailed Description	37
6.18	MD5_MB_MGR_X8X2 Struct Reference	37
6.18.1	Detailed Description	38
6.19	SHA1_ARGS_X4 Struct Reference	38
6.19.1	Detailed Description	38
6.20	SHA1_ARGS_X8 Struct Reference	38
6.20.1	Detailed Description	39
6.21	SHA1_HASH_CTX Struct Reference	39
6.21.1	Detailed Description	39
6.22	SHA1_HASH_CTX_MGR Struct Reference	40
6.22.1	Detailed Description	40
6.23	SHA1_HMAC_LANE_DATA Struct Reference	40
6.23.1	Detailed Description	40
6.24	SHA1_JOB Struct Reference	41
6.24.1	Detailed Description	41
6.25	SHA1_LANE_DATA Struct Reference	41
6.25.1	Detailed Description	41
6.26	SHA1_MB_ARGS_X8 Struct Reference	42
6.26.1	Detailed Description	42
6.27	SHA1_MB_JOB_MGR Struct Reference	42
6.27.1	Detailed Description	42
6.28	SHA1_MB_MGR Struct Reference	42
6.28.1	Detailed Description	43
6.29	SHA1_MB_MGR_X8 Struct Reference	43
6.29.1	Detailed Description	44
6.30	SHA256_ARGS_X4 Struct Reference	44

---

6.30.1 Detailed Description . . . . .	44
6.31 SHA256_ARGS_X8 Struct Reference . . . . .	44
6.31.1 Detailed Description . . . . .	45
6.32 SHA256_HASH_CTX Struct Reference . . . . .	45
6.32.1 Detailed Description . . . . .	45
6.33 SHA256_HASH_CTX_MGR Struct Reference . . . . .	46
6.33.1 Detailed Description . . . . .	46
6.34 SHA256_HMAC_LANE_DATA Struct Reference . . . . .	46
6.34.1 Detailed Description . . . . .	46
6.35 SHA256_JOB Struct Reference . . . . .	47
6.35.1 Detailed Description . . . . .	47
6.36 SHA256_LANE_DATA Struct Reference . . . . .	47
6.36.1 Detailed Description . . . . .	47
6.37 SHA256_MB_ARGS_X8 Struct Reference . . . . .	48
6.37.1 Detailed Description . . . . .	48
6.38 SHA256_MB_JOB_MGR Struct Reference . . . . .	48
6.38.1 Detailed Description . . . . .	48
6.39 SHA256_MB_MGR Struct Reference . . . . .	48
6.39.1 Detailed Description . . . . .	49
6.40 SHA256_MB_MGR_X8 Struct Reference . . . . .	49
6.40.1 Detailed Description . . . . .	49
6.41 SHA512_ARGS_X2 Struct Reference . . . . .	50
6.41.1 Detailed Description . . . . .	50
6.42 SHA512_ARGS_X4 Struct Reference . . . . .	50
6.42.1 Detailed Description . . . . .	50
6.43 SHA512_HASH_CTX Struct Reference . . . . .	51
6.43.1 Detailed Description . . . . .	51
6.44 SHA512_HASH_CTX_MGR Struct Reference . . . . .	51
6.44.1 Detailed Description . . . . .	51
6.45 SHA512_HMAC_LANE_DATA Struct Reference . . . . .	52
6.45.1 Detailed Description . . . . .	52
6.46 SHA512_JOB Struct Reference . . . . .	52
6.46.1 Detailed Description . . . . .	53
6.47 SHA512_LANE_DATA Struct Reference . . . . .	53
6.47.1 Detailed Description . . . . .	53
6.48 SHA512_MB_ARGS_X4 Struct Reference . . . . .	53
6.48.1 Detailed Description . . . . .	54
6.49 SHA512_MB_JOB_MGR Struct Reference . . . . .	54
6.49.1 Detailed Description . . . . .	54
6.50 SHA512_MB_MGR Struct Reference . . . . .	54
6.50.1 Detailed Description . . . . .	55
6.51 SHA512_MB_MGR_X4 Struct Reference . . . . .	55
6.51.1 Detailed Description . . . . .	55
<b>7 File Documentation . . . . .</b>	<b>56</b>
7.1 aes_xts.h File Reference . . . . .	56

---

7.1.1	Detailed Description	57
7.1.2	Function Documentation	58
7.1.2.1	aes_keyexp_128	58
7.1.2.2	aes_keyexp_256	58
7.1.2.3	XTS_AES_128_dec	58
7.1.2.4	XTS_AES_128_dec_expanded_key	59
7.1.2.5	XTS_AES_128_enc	59
7.1.2.6	XTS_AES_128_enc_expanded_key	60
7.1.2.7	XTS_AES_256_dec	60
7.1.2.8	XTS_AES_256_dec_expanded_key	60
7.1.2.9	XTS_AES_256_enc	61
7.1.2.10	XTS_AES_256_enc_expanded_key	61
7.2	crc.h File Reference	62
7.2.1	Detailed Description	62
7.2.2	Function Documentation	63
7.2.2.1	crc16_t10dif	63
7.2.2.2	crc16_t10dif_01	63
7.2.2.3	crc16_t10dif_base	63
7.2.2.4	crc16_t10dif_by4	64
7.2.2.5	crc32_ieee	64
7.2.2.6	crc32_ieee_01	65
7.2.2.7	crc32_ieee_base	65
7.2.2.8	crc32_ieee_by4	65
7.2.2.9	crc32_iscsi	66
7.2.2.10	crc32_iscsi_00	66
7.2.2.11	crc32_iscsi_01	67
7.2.2.12	crc32_iscsi_base	67
7.2.2.13	crc32_iscsi_baseline	67
7.2.2.14	crc32_iscsi_simple	68
7.3	erasure_code.h File Reference	68
7.3.1	Detailed Description	72
7.3.2	Function Documentation	72
7.3.2.1	ec_encode_data	72
7.3.2.2	ec_encode_data_avx	73
7.3.2.3	ec_encode_data_avx2	73
7.3.2.4	ec_encode_data_base	73
7.3.2.5	ec_encode_data_sse	74
7.3.2.6	ec_encode_data_update	74
7.3.2.7	ec_encode_data_update_avx	74
7.3.2.8	ec_encode_data_update_avx2	75
7.3.2.9	ec_encode_data_update_base	75
7.3.2.10	ec_encode_data_update_sse	75
7.3.2.11	ec_init_tables	75
7.3.2.12	gf_2vect_dot_prod_avx	76
7.3.2.13	gf_2vect_dot_prod_avx2	76
7.3.2.14	gf_2vect_dot_prod_sse	77

---

7.3.2.15	gf_2vect_mad_avx	77
7.3.2.16	gf_2vect_mad_avx2	77
7.3.2.17	gf_2vect_mad_sse	78
7.3.2.18	gf_3vect_dot_prod_avx	78
7.3.2.19	gf_3vect_dot_prod_avx2	79
7.3.2.20	gf_3vect_dot_prod_sse	79
7.3.2.21	gf_3vect_mad_avx	80
7.3.2.22	gf_3vect_mad_avx2	80
7.3.2.23	gf_3vect_mad_sse	80
7.3.2.24	gf_4vect_dot_prod_avx	81
7.3.2.25	gf_4vect_dot_prod_avx2	81
7.3.2.26	gf_4vect_dot_prod_sse	82
7.3.2.27	gf_4vect_mad_avx	83
7.3.2.28	gf_4vect_mad_avx2	83
7.3.2.29	gf_4vect_mad_sse	83
7.3.2.30	gf_5vect_dot_prod_avx	84
7.3.2.31	gf_5vect_dot_prod_avx2	84
7.3.2.32	gf_5vect_dot_prod_sse	85
7.3.2.33	gf_5vect_mad_avx	85
7.3.2.34	gf_5vect_mad_avx2	86
7.3.2.35	gf_5vect_mad_sse	86
7.3.2.36	gf_6vect_dot_prod_avx	86
7.3.2.37	gf_6vect_dot_prod_avx2	87
7.3.2.38	gf_6vect_dot_prod_sse	87
7.3.2.39	gf_6vect_mad_avx	88
7.3.2.40	gf_6vect_mad_avx2	88
7.3.2.41	gf_6vect_mad_sse	88
7.3.2.42	gf_gen_cauchy1_matrix	88
7.3.2.43	gf_gen_rs_matrix	89
7.3.2.44	gf_inv	89
7.3.2.45	gf_invert_matrix	90
7.3.2.46	gf_mul	90
7.3.2.47	gf_vect_dot_prod	90
7.3.2.48	gf_vect_dot_prod_avx	91
7.3.2.49	gf_vect_dot_prod_avx2	91
7.3.2.50	gf_vect_dot_prod_base	92
7.3.2.51	gf_vect_dot_prod_sse	92
7.3.2.52	gf_vect_mad	93
7.3.2.53	gf_vect_mad_avx	93
7.3.2.54	gf_vect_mad_avx2	93
7.3.2.55	gf_vect_mad_base	94
7.3.2.56	gf_vect_mad_sse	94
7.4	gf_vect_mul.h File Reference	94
7.4.1	Detailed Description	94
7.4.2	Function Documentation	95
7.4.2.1	gf_vect_mul	95

7.4.2.2	gf_vect_mul_avx	95
7.4.2.3	gf_vect_mul_base	96
7.4.2.4	gf_vect_mul_init	96
7.4.2.5	gf_vect_mul_sse	96
7.5	igzip_lib.h File Reference	97
7.5.1	Detailed Description	98
7.5.2	Enumeration Type Documentation	99
7.5.2.1	LZ_State1_state	99
7.5.3	Function Documentation	99
7.5.3.1	fast_lz	99
7.5.3.2	fast_lz_stateless	100
7.5.3.3	init_stream	101
7.6	intrinreg.h File Reference	101
7.6.1	Detailed Description	101
7.7	mb_md5.h File Reference	101
7.7.1	Detailed Description	102
7.7.2	Function Documentation	103
7.7.2.1	md5_flush_job	103
7.7.2.2	md5_flush_job_avx	103
7.7.2.3	md5_flush_job_avx2	103
7.7.2.4	md5_init_mb_mgr	104
7.7.2.5	md5_init_mb_mgr_x8x2	104
7.7.2.6	md5_submit_job	104
7.7.2.7	md5_submit_job_avx	105
7.7.2.8	md5_submit_job_avx2	105
7.8	mb_shal.h File Reference	106
7.8.1	Detailed Description	107
7.8.2	Function Documentation	107
7.8.2.1	shal_flush_job	107
7.8.2.2	shal_flush_job_avx	107
7.8.2.3	shal_flush_job_avx2	108
7.8.2.4	shal_init_mb_mgr	108
7.8.2.5	shal_init_mb_mgr_x8	109
7.8.2.6	shal_submit_job	109
7.8.2.7	shal_submit_job_avx	109
7.8.2.8	shal_submit_job_avx2	110
7.9	mb_sha256.h File Reference	110
7.9.1	Detailed Description	111
7.9.2	Function Documentation	112
7.9.2.1	sha256_flush_job	112
7.9.2.2	sha256_flush_job_avx	112
7.9.2.3	sha256_flush_job_avx2	112
7.9.2.4	sha256_init_mb_mgr	113
7.9.2.5	sha256_init_mb_mgr_x8	113
7.9.2.6	sha256_submit_job	113
7.9.2.7	sha256_submit_job_avx	114



7.9.2.8	sha256_submit_job_avx2	114
7.10	mb_sha512.h File Reference	115
7.10.1	Detailed Description	116
7.10.2	Function Documentation	116
7.10.2.1	sha512_flush_job	116
7.10.2.2	sha512_flush_job_avx	116
7.10.2.3	sha512_flush_job_avx2	117
7.10.2.4	sha512_init_mb_mgr	117
7.10.2.5	sha512_init_mb_mgr_x4	117
7.10.2.6	sha512_submit_job	118
7.10.2.7	sha512_submit_job_avx	118
7.10.2.8	sha512_submit_job_avx2	118
7.11	md5_mb.h File Reference	119
7.11.1	Detailed Description	120
7.11.2	Function Documentation	121
7.11.2.1	md5_ctx_mgr_flush	121
7.11.2.2	md5_ctx_mgr_flush_avx	122
7.11.2.3	md5_ctx_mgr_flush_avx2	122
7.11.2.4	md5_ctx_mgr_flush_sse	122
7.11.2.5	md5_ctx_mgr_init	123
7.11.2.6	md5_ctx_mgr_init_avx	123
7.11.2.7	md5_ctx_mgr_init_avx2	123
7.11.2.8	md5_ctx_mgr_init_sse	124
7.11.2.9	md5_ctx_mgr_submit	124
7.11.2.10	md5_ctx_mgr_submit_avx	125
7.11.2.11	md5_ctx_mgr_submit_avx2	125
7.11.2.12	md5_ctx_mgr_submit_sse	125
7.12	mem_routines.h File Reference	126
7.12.1	Detailed Description	126
7.12.2	Function Documentation	127
7.12.2.1	mem_cmp_avx	127
7.12.2.2	mem_cmp_avx2	127
7.12.2.3	mem_cmp_sse	127
7.12.2.4	mem_cpy_avx	128
7.12.2.5	mem_cpy_sse	128
7.12.2.6	mem_zero_detect_avx	129
7.13	memcpy_inline.h File Reference	129
7.13.1	Detailed Description	129
7.14	multi_buffer.h File Reference	129
7.14.1	Detailed Description	130
7.14.2	Enumeration Type Documentation	130
7.14.2.1	HASH_CTX_ERROR	130
7.14.2.2	HASH_CTX_FLAG	130
7.14.2.3	HASH_CTX_STS	131
7.14.2.4	JOB_STS	131
7.15	raid.h File Reference	131

---

7.15.1	Detailed Description	132
7.15.2	Function Documentation	132
7.15.2.1	pq_check	132
7.15.2.2	pq_check_base	133
7.15.2.3	pq_check_sse	133
7.15.2.4	pq_gen	133
7.15.2.5	pq_gen_avx	134
7.15.2.6	pq_gen_avx2	134
7.15.2.7	pq_gen_base	135
7.15.2.8	pq_gen_sse	135
7.15.2.9	xor_check	135
7.15.2.10	xor_check_base	136
7.15.2.11	xor_check_sse	136
7.15.2.12	xor_gen	137
7.15.2.13	xor_gen_avx	137
7.15.2.14	xor_gen_base	137
7.15.2.15	xor_gen_sse	138
7.16	sha.h File Reference	138
7.16.1	Detailed Description	138
7.16.2	Function Documentation	139
7.16.2.1	sha1_opt	139
7.16.2.2	sha1_update	139
7.17	sha1_mb.h File Reference	139
7.17.1	Detailed Description	141
7.17.2	Function Documentation	142
7.17.2.1	sha1_ctx_mgr_flush	142
7.17.2.2	sha1_ctx_mgr_flush_avx	142
7.17.2.3	sha1_ctx_mgr_flush_avx2	143
7.17.2.4	sha1_ctx_mgr_flush_sse	143
7.17.2.5	sha1_ctx_mgr_init	143
7.17.2.6	sha1_ctx_mgr_init_avx	144
7.17.2.7	sha1_ctx_mgr_init_avx2	144
7.17.2.8	sha1_ctx_mgr_init_sse	144
7.17.2.9	sha1_ctx_mgr_submit	145
7.17.2.10	sha1_ctx_mgr_submit_avx	145
7.17.2.11	sha1_ctx_mgr_submit_avx2	146
7.17.2.12	sha1_ctx_mgr_submit_sse	146
7.18	sha256_mb.h File Reference	147
7.18.1	Detailed Description	148
7.18.2	Function Documentation	149
7.18.2.1	sha256_ctx_mgr_flush	149
7.18.2.2	sha256_ctx_mgr_flush_avx	150
7.18.2.3	sha256_ctx_mgr_flush_avx2	150
7.18.2.4	sha256_ctx_mgr_flush_sse	150
7.18.2.5	sha256_ctx_mgr_init	151
7.18.2.6	sha256_ctx_mgr_init_avx	151

---

7.18.2.7	sha256_ctx_mgr_init_avx2	151
7.18.2.8	sha256_ctx_mgr_init_sse	152
7.18.2.9	sha256_ctx_mgr_submit	152
7.18.2.10	sha256_ctx_mgr_submit_avx	153
7.18.2.11	sha256_ctx_mgr_submit_avx2	153
7.18.2.12	sha256_ctx_mgr_submit_sse	153
7.19	sha512_mb.h File Reference	154
7.19.1	Detailed Description	156
7.19.2	Function Documentation	157
7.19.2.1	sha512_ctx_mgr_flush	157
7.19.2.2	sha512_ctx_mgr_flush_avx	157
7.19.2.3	sha512_ctx_mgr_flush_avx2	158
7.19.2.4	sha512_ctx_mgr_flush_sb_sse4	158
7.19.2.5	sha512_ctx_mgr_flush_sse	158
7.19.2.6	sha512_ctx_mgr_init	159
7.19.2.7	sha512_ctx_mgr_init_avx	159
7.19.2.8	sha512_ctx_mgr_init_avx2	159
7.19.2.9	sha512_ctx_mgr_init_sb_sse4	160
7.19.2.10	sha512_ctx_mgr_init_sse	160
7.19.2.11	sha512_ctx_mgr_submit	160
7.19.2.12	sha512_ctx_mgr_submit_avx	161
7.19.2.13	sha512_ctx_mgr_submit_avx2	161
7.19.2.14	sha512_ctx_mgr_submit_sb_sse4	162
7.19.2.15	sha512_ctx_mgr_submit_sse	162
7.20	types.h File Reference	163
7.20.1	Detailed Description	163
<b>8</b>	<b>Example Documentation</b>	<b>164</b>
8.1	crc_simple_test.c	164
8.2	igzip_example.c	165
8.3	multi_buffer_sha1_example.c	166
8.4	xor_example.c	168
<b>Index</b>		<b>170</b>

### 1.1 About This Document

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

### 1.2 Overview

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

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

### 1.3 RAID Functions

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

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

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

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

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

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

## 1.4 Erasure Code Functions

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

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

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

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

## 1.5 CRC Functions

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

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

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

## 1.6 Multi-buffer Hashing Functions

Functions in the Multi-buffer MD5, SHA1, SHA256 and SHA512 sections are used to increase the performance of the secure hash algorithms on a single processor core by operating on multiple jobs at once. By buffering jobs, the algorithm can exploit the instruction-level parallelism inherent in modern IA cores to an extent not possible in a serial implementation. The multi-buffer API is similar to that used in the [whitepaper Fast Multi-buffer IPsec Implementations on Intel Architecture Processors](#).

There are two flavors of the multi-buffer hash API: the older version that uses the MB\_MGR structure context and the newer version that uses the HASH\_CTX\_MGR context (MD5 is not available under the new API at this time). Changes to the API were necessary to get new functionality. Benefits to using the new API include:

- Multibinary functionality. Call one function and the appropriate architecture-specific version is fixed up at runtime.

- No restriction on update length. Submitting an update block no longer has to have length a multiple of the fundamental block size.

As noted in the above document, the scheduler routines do not enforce atomic access to the context structure. If a single scheduler state structure is being used by multiple threads, then the application must take care that calls are not made from different threads at the same time, i.e. thread-safety should be implemented at a level higher than these routines. This could be implemented by employing a separate context structure for each worker thread.

## 1.7 Alignment for Input Parameters

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

Function	Alignment Required
AES-XTS 128	No
AES-XTS 256	No
CRC	No
Erasur Code	32B for gf_vect_mul, none otherwise
RAID	32B or 16B
Igzip	No
MB Hashing - old API	No (Members of JOB structures defined with required alignment, already aligned once initialised. On FreeBSD the MB_MGR structure may need to be aligned to 32B for AVX2 MD5)
MB Hashing - new API	No (Members of CTX structures defined with required alignment, already aligned once initialised. On FreeBSD, or when using Linux/icc, the CTX_MGR structure may need to be aligned to 16B.)

## 1.8 System Requirements

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

### Recommended Hardware:

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

### Software Requirements:

Most functions in the library use the 64-bit embedded and Unix standard for calling convention [http://refspecs.linuxfoundation.org/elf/x86\\_64-abi-0.95.pdf](http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.95.pdf). When available, 32-bit versions use cdecl. Individual functions are written to be statically linked with an application.

### Building Library Functions:

- Yasm Assembler: version at least v1.2.0.

### Building Examples and Tests:

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

### Note

Please note that the library assumes  $1\text{MB} = 1,000,000$  bytes in reported performance figures.

---

## CHAPTER 2

# FUNCTION VERSION NUMBERS

---

### 2.1 Function Version Numbers

Individual functions are given version numbers with the format mm-vv-ssss.

- mm = Two hex digits indicating the processor a function was optimized for.

- 00 = Nehalem/Jasper Forest/Multibinary
- 01 = Westmere
- 02 = Sandybridge
- 03 = Ivy Bridge
- 04 = Haswell
- 05 = Silvermont

- vv = function version number

- ssss = function serial number



## 2.2 Function Version Numbers Tables

Function	Version
<a href="#">crc16_t10dif_01</a>	01-05-0010
<a href="#">crc32_ieee_01</a>	01-05-0011
<a href="#">crc32_iscsi_simple</a>	00-01-0012
<a href="#">crc32_iscsi_baseline</a>	00-01-0013
<a href="#">crc32_iscsi_00</a>	00-02-0014
<a href="#">crc32_iscsi_01</a>	01-02-0015
<a href="#">crc16_t10dif_by4</a>	05-01-0016
<a href="#">crc32_ieee_by4</a>	05-01-0017
<a href="#">sha1_init_mb_mgr</a>	00-03-0020
<a href="#">sha1_flush_job</a>	00-07-0021
<a href="#">sha1_submit_job</a>	00-07-0022
<a href="#">sha256_init_mb_mgr</a>	00-03-0023
<a href="#">sha256_flush_job</a>	00-07-0024
<a href="#">sha256_submit_job</a>	00-07-0025
<a href="#">md5_init_mb_mgr</a>	00-02-0026
<a href="#">md5_flush_job</a>	00-06-0027
<a href="#">md5_submit_job</a>	00-06-0028
<a href="#">sha512_init_mb_mgr</a>	00-05-002a
<a href="#">sha512_flush_job</a>	00-06-002b
<a href="#">sha512_submit_job</a>	00-06-002c
<a href="#">xor_gen_sse</a>	00-0b-0030
<a href="#">xor_check_sse</a>	00-02-0031
<a href="#">pq_gen_sse</a>	00-08-0032
<a href="#">pq_check_sse</a>	00-05-0033
<a href="#">gf_vect_mul_sse</a>	00-02-0034
<a href="#">gf_vect_mul_init</a>	00-02-0035
<a href="#">gf_vect_mul_avx</a>	01-02-0036
<a href="#">xor_gen_avx</a>	02-04-0037
<a href="#">pq_gen_avx</a>	02-09-0039
<a href="#">pq_gen_avx2</a>	04-02-0041
<a href="#">sha1_update</a>	00-01-0050
<a href="#">sha1_opt</a>	00-02-0051
<a href="#">gf_vect_dot_prod_sse</a>	00-03-0060
<a href="#">gf_vect_dot_prod_avx</a>	02-03-0061
<a href="#">gf_2vect_dot_prod_sse</a>	00-02-0062
<a href="#">gf_3vect_dot_prod_sse</a>	00-03-0063
<a href="#">gf_4vect_dot_prod_sse</a>	00-03-0064
<a href="#">gf_5vect_dot_prod_sse</a>	00-03-0065
<a href="#">gf_6vect_dot_prod_sse</a>	00-03-0066
<a href="#">ec_init_tables</a>	00-01-0068
<a href="#">ec_encode_data_sse</a>	00-02-0069

Function	Version
aes_keyexp_128	01-02-0070
XTS_AES_128_enc	01-03-0071
XTS_AES_128_enc_expanded_key	01-03-0072
XTS_AES_128_dec	01-03-0073
XTS_AES_128_dec_expanded_key	01-03-0074
aes_keyexp_256	01-02-0075
XTS_AES_256_enc	01-03-0076
XTS_AES_256_enc_expanded_key	01-03-0077
XTS_AES_256_dec	01-03-0078
XTS_AES_256_dec_expanded_key	01-03-0079
init_stream	01-03-0081
fast_lz	01-03-0082
fast_lz_stateless	01-01-0083
sha1_flush_job_avx	02-07-0091
sha1_submit_job_avx	02-07-0092
sha256_flush_job_avx	02-07-0094
sha256_submit_job_avx	02-07-0095
md5_flush_job_avx	02-06-0097
md5_submit_job_avx	02-06-0098
sha512_flush_job_avx	02-06-009b
sha512_submit_job_avx	02-06-009c
sha1_init_mb_mgr_x8	04-01-0100
sha1_flush_job_avx2	04-01-0101
sha1_submit_job_avx2	04-01-0102
sha256_init_mb_mgr_x8	04-01-0103
sha256_flush_job_avx2	04-01-0104
sha256_submit_job_avx2	04-01-0105
md5_init_mb_mgr_x8x2	04-01-0106
md5_submit_job_avx2	04-01-0107
md5_flush_job_avx2	04-01-0108
sha512_init_mb_mgr_x4	04-01-0109
sha512_submit_job_avx2	04-01-0110
sha512_flush_job_avx2	04-01-0111
crc16_t10dif	00-02-011a
crc32_ieee	00-02-011b
crc32_iscsi	00-02-011c
crc32_iscsi_base	00-01-011d
crc16_t10dif_base	00-01-011e
crc32_ieee_base	00-01-011f
xor_gen	00-02-0126

Function	Version
xor_check	00-02-0127
pq_gen	00-02-0128
pq_check	00-02-0129
pq_gen_base	00-01-012a
xor_gen_base	00-01-012b
pq_check_base	00-01-012c
xor_check_base	00-01-012d
ec_encode_data	00-02-0133
gf_vect_mul	00-02-0134
ec_encode_data_base	00-01-0135
gf_vect_mul_base	00-01-0136
gf_vect_dot_prod_base	00-01-0137
gf_vect_dot_prod	00-01-0138
sha1_ctx_mgr_init_sse	00-01-0139
sha1_ctx_mgr_submit_sse	00-01-0140
sha1_ctx_mgr_flush_sse	00-01-0141
sha1_ctx_mgr_init_avx	02-01-0142
sha1_ctx_mgr_submit_avx	02-01-0143
sha1_ctx_mgr_flush_avx	02-01-0144
sha1_ctx_mgr_init_avx2	04-01-0145
sha1_ctx_mgr_submit_avx2	04-01-0146
sha1_ctx_mgr_flush_avx2	04-01-0147
sha1_ctx_mgr_init	00-01-0148
sha1_ctx_mgr_submit	00-01-0149
sha1_ctx_mgr_flush	00-01-0150
sha256_ctx_mgr_init_sse	00-01-0151
sha256_ctx_mgr_submit_sse	00-01-0152
sha256_ctx_mgr_flush_sse	00-01-0153
sha256_ctx_mgr_init_avx	02-01-0154
sha256_ctx_mgr_submit_avx	02-01-0155
sha256_ctx_mgr_flush_avx	02-01-0156
sha256_ctx_mgr_init_avx2	04-01-0157
sha256_ctx_mgr_submit_avx2	04-01-0158
sha256_ctx_mgr_flush_avx2	04-01-0159
sha256_ctx_mgr_init	00-01-0160
sha256_ctx_mgr_submit	00-01-0161
sha256_ctx_mgr_flush	00-01-0162
sha512_ctx_mgr_init_sse	00-01-0163
sha512_ctx_mgr_submit_sse	00-01-0164
sha512_ctx_mgr_flush_sse	00-01-0165

Function	Version
sha512_ctx_mgr_init_avx	02-01-0166
sha512_ctx_mgr_submit_avx	02-01-0167
sha512_ctx_mgr_flush_avx	02-01-0168
sha512_ctx_mgr_init_avx2	04-01-0169
sha512_ctx_mgr_submit_avx2	04-01-0170
sha512_ctx_mgr_flush_avx2	04-01-0171
sha512_ctx_mgr_init_sb_sse4	05-01-0172
sha512_ctx_mgr_submit_sb_sse4	05-01-0173
sha512_ctx_mgr_flush_sb_sse4	05-01-0174
sha512_ctx_mgr_init	00-01-0175
sha512_ctx_mgr_submit	00-01-0176
sha512_ctx_mgr_flush	00-01-0177
md5_ctx_mgr_init_sse	00-01-0180
md5_ctx_mgr_submit_sse	00-01-0181
md5_ctx_mgr_flush_sse	00-01-0182
md5_ctx_mgr_init_avx	02-01-0183
md5_ctx_mgr_submit_avx	02-01-0184
md5_ctx_mgr_flush_avx	02-01-0185
md5_ctx_mgr_init_avx2	04-01-0186
md5_ctx_mgr_submit_avx2	04-01-0187
md5_ctx_mgr_flush_avx2	04-01-0188
md5_ctx_mgr_init	00-01-0189
md5_ctx_mgr_submit	00-01-018a
md5_ctx_mgr_flush	00-01-018b
gf_vect_dot_prod_avx2	04-03-0190
gf_2vect_dot_prod_avx	02-03-0191
gf_3vect_dot_prod_avx	02-03-0192
gf_4vect_dot_prod_avx	02-02-0193
gf_5vect_dot_prod_avx	02-03-0194
gf_6vect_dot_prod_avx	02-03-0195
gf_2vect_dot_prod_avx2	04-03-0196
gf_3vect_dot_prod_avx2	04-03-0197
gf_4vect_dot_prod_avx2	04-03-0198
gf_5vect_dot_prod_avx2	04-03-0199
gf_6vect_dot_prod_avx2	04-03-019a
gf_vect_mad_sse	00-00-0200
gf_vect_mad_avx	02-00-0201
gf_vect_mad_avx2	04-00-0202
gf_2vect_mad_sse	00-00-0203
gf_2vect_mad_avx	02-00-0204

Function	Version
gf_2vect_mad_avx2	04-00-0205
gf_3vect_mad_sse	00-00-0206
gf_3vect_mad_avx	02-00-0207
gf_3vect_mad_avx2	04-00-0208
gf_4vect_mad_sse	00-00-0209
gf_4vect_mad_avx	02-00-020a
gf_4vect_mad_avx2	04-00-020b
gf_5vect_mad_sse	00-00-020c
gf_5vect_mad_avx	02-00-020d
gf_5vect_mad_avx2	04-00-020e
gf_6vect_mad_sse	00-00-020f
gf_6vect_mad_avx	02-00-0210
gf_6vect_mad_avx2	04-00-0211
ec_encode_data_update	00-02-0212
gf_vect_mad	00-01-0213
mem_zero_detect_avx	02-01-0232
mem_cpy_sse	02-01-0236
mem_cpy_avx	02-01-0237
mem_cmp_sse	02-01-0241
mem_cmp_avx	02-01-0242
mem_cmp_avx2	02-01-0243

## CHAPTER 3

# INSTRUCTION SET REQUIREMENTS

---

[aes\\_keyexp\\_128](#) (UINT8 \*key, UINT8 \*exp\_key\_enc, UINT8 \*exp\_key\_dec)  
AES-NI

[aes\\_keyexp\\_256](#) (UINT8 \*key, UINT8 \*exp\_key\_enc, UINT8 \*exp\_key\_dec)  
AES-NI

[crc16\\_t10dif\\_01](#) (UINT16 init\_crc, const unsigned char \*buf, UINT64 len)  
SSE3, CLMUL

[crc16\\_t10dif\\_by4](#) (UINT16 init\_crc, const unsigned char \*buf, UINT64 len)  
SSE4, PCLMULQDQ.

[crc32\\_ieee\\_01](#) (UINT32 init\_crc, const unsigned char \*buf, UINT64 len)  
SSE3, CLMUL

[crc32\\_ieee\\_by4](#) (UINT32 init\_crc, const unsigned char \*buf, UINT64 len)  
SSE4, PCLMULQDQ.

[crc32\\_iscsi\\_00](#) (unsigned char \*buffer, int len, unsigned int init\_crc)  
SSE4.2

[crc32\\_iscsi\\_01](#) (unsigned char \*buffer, int len, unsigned int init\_crc)  
SSE4.2, CLMUL

[crc32\\_iscsi\\_baseline](#) (unsigned char \*buffer, int len, unsigned int init\_crc)  
SSE4.2

[crc32\\_iscsi\\_simple](#) (unsigned char \*buffer, int len, unsigned int init\_crc)  
SSE4.2

[ec\\_encode\\_data\\_avx](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
AVX

[ec\\_encode\\_data\\_avx2](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
AVX2

[ec\\_encode\\_data\\_sse](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
SSE4.1

[ec\\_encode\\_data\\_update\\_avx](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
AVX

[ec\\_encode\\_data\\_update\\_avx2](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
AVX2

---

`ec_encode_data_update_sse` (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
SSE4.1

`fast_lz` (LZ\_Stream1 \*stream)  
SSE4.1, CLMUL

`fast_lz_stateless` (LZ\_Stream1 \*stream)  
SSE4.1, CLMUL

`gf_2vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_2vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_2vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_2vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_2vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_2vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_3vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_3vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_3vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_3vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_3vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_3vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_4vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_4vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

---

---

`gf_4vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_4vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_4vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_4vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_5vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_5vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_5vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_5vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_5vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_5vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_6vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_6vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_6vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_6vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_6vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_6vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
AVX

`gf_vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
AVX2

---



---

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

`gf_vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
AVX

`gf_vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
AVX2

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

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

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

`init_stream` (LZ\_Stream1 \*stream)  
SSE4.1, CLMUL

`md5_ctx_mgr_flush` (MD5\_HASH\_CTX\_MGR \*mgr)  
SSE4.1 or AVX or AVX2

`md5_ctx_mgr_flush_avx` (MD5\_HASH\_CTX\_MGR \*mgr)  
AVX

`md5_ctx_mgr_flush_avx2` (MD5\_HASH\_CTX\_MGR \*mgr)  
AVX2

`md5_ctx_mgr_flush_sse` (MD5\_HASH\_CTX\_MGR \*mgr)  
SSE4.1

`md5_ctx_mgr_init` (MD5\_HASH\_CTX\_MGR \*mgr)  
SSE4.1 or AVX or AVX2

`md5_ctx_mgr_init_avx` (MD5\_HASH\_CTX\_MGR \*mgr)  
AVX

`md5_ctx_mgr_init_avx2` (MD5\_HASH\_CTX\_MGR \*mgr)  
AVX2

`md5_ctx_mgr_init_sse` (MD5\_HASH\_CTX\_MGR \*mgr)  
SSE4.1

`md5_ctx_mgr_submit` (MD5\_HASH\_CTX\_MGR \*mgr, MD5\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4.1 or AVX or AVX2

`md5_ctx_mgr_submit_avx` (MD5\_HASH\_CTX\_MGR \*mgr, MD5\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
AVX

---

---

`md5_ctx_mgr_submit_avx2` (`MD5_HASH_CTX_MGR` \*mgr, `MD5_HASH_CTX` \*ctx, const void \*buffer, uint32\_t len, `HASH_CTX_FLAG` flags)  
AVX2

`md5_ctx_mgr_submit_sse` (`MD5_HASH_CTX_MGR` \*mgr, `MD5_HASH_CTX` \*ctx, const void \*buffer, uint32\_t len, `HASH_CTX_FLAG` flags)  
SSE4.1

`md5_flush_job` (`MD5_MB_MGR` \*state)  
SSE4.1

`md5_flush_job_avx` (`MD5_MB_MGR` \*state)  
AVX

`md5_flush_job_avx2` (`MD5_MB_MGR_X8X2` \*state)  
AVX2

`md5_init_mb_mgr` (`MD5_MB_MGR` \*state)  
SSE4.1

`md5_init_mb_mgr_x8x2` (`MD5_MB_MGR_X8X2` \*state)  
AVX2

`md5_submit_job` (`MD5_MB_MGR` \*state, `JOB_MD5` \*job)  
SSE4.1

`md5_submit_job_avx` (`MD5_MB_MGR` \*state, `JOB_MD5` \*job)  
AVX

`md5_submit_job_avx2` (`MD5_MB_MGR_X8X2` \*state, `JOB_MD5` \*job)  
AVX2

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

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

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

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

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

`mem_zero_detect_avx` (void \*mem, int len)  
AVX

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

---

---

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

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

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

`sha1_ctx_mgr_flush` (SHA1\_HASH\_CTX\_MGR \*mgr)  
SSE4.1 or AVX or AVX2

`sha1_ctx_mgr_flush_avx` (SHA1\_HASH\_CTX\_MGR \*mgr)  
AVX

`sha1_ctx_mgr_flush_avx2` (SHA1\_HASH\_CTX\_MGR \*mgr)  
AVX2

`sha1_ctx_mgr_flush_sse` (SHA1\_HASH\_CTX\_MGR \*mgr)  
SSE4.1

`sha1_ctx_mgr_init` (SHA1\_HASH\_CTX\_MGR \*mgr)  
SSE4.1 or AVX or AVX2

`sha1_ctx_mgr_init_avx` (SHA1\_HASH\_CTX\_MGR \*mgr)  
AVX

`sha1_ctx_mgr_init_avx2` (SHA1\_HASH\_CTX\_MGR \*mgr)  
AVX2

`sha1_ctx_mgr_init_sse` (SHA1\_HASH\_CTX\_MGR \*mgr)  
SSE4.1

`sha1_ctx_mgr_submit` (SHA1\_HASH\_CTX\_MGR \*mgr, SHA1\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4.1 or AVX or AVX2

`sha1_ctx_mgr_submit_avx` (SHA1\_HASH\_CTX\_MGR \*mgr, SHA1\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
AVX

`sha1_ctx_mgr_submit_avx2` (SHA1\_HASH\_CTX\_MGR \*mgr, SHA1\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
AVX2

`sha1_ctx_mgr_submit_sse` (SHA1\_HASH\_CTX\_MGR \*mgr, SHA1\_HASH\_CTX \*ctx, const void \*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4.1

`sha1_flush_job` (SHA1\_MB\_MGR \*state)  
SSE4.1

`sha1_flush_job_avx` (SHA1\_MB\_MGR \*state)  
AVX

---

---

`sha1_flush_job_avx2 (SHA1_MB_MGR_X8 *state)`  
AVX2

`sha1_init_mb_mgr (SHA1_MB_MGR *state)`  
SSE4.1

`sha1_init_mb_mgr_x8 (SHA1_MB_MGR_X8 *state)`  
AVX2

`sha1_opt (unsigned char *input, unsigned int *digest, int len)`  
SSE3

`sha1_submit_job (SHA1_MB_MGR *state, JOB_SHA1 *job)`  
SSE4.1

`sha1_submit_job_avx (SHA1_MB_MGR *state, JOB_SHA1 *job)`  
AVX

`sha1_submit_job_avx2 (SHA1_MB_MGR_X8 *state, JOB_SHA1 *job)`  
AVX2

`sha1_update (unsigned int *digest, unsigned char *input, size_t num_blocks)`  
SSE3

`sha256_ctx_mgr_flush (SHA256_HASH_CTX_MGR *mgr)`  
SSE4.1 or AVX or AVX2

`sha256_ctx_mgr_flush_avx (SHA256_HASH_CTX_MGR *mgr)`  
AVX

`sha256_ctx_mgr_flush_avx2 (SHA256_HASH_CTX_MGR *mgr)`  
AVX2

`sha256_ctx_mgr_flush_sse (SHA256_HASH_CTX_MGR *mgr)`  
SSE4.1

`sha256_ctx_mgr_init (SHA256_HASH_CTX_MGR *mgr)`  
SSE4.1 or AVX or AVX2

`sha256_ctx_mgr_init_avx (SHA256_HASH_CTX_MGR *mgr)`  
AVX

`sha256_ctx_mgr_init_avx2 (SHA256_HASH_CTX_MGR *mgr)`  
AVX2

`sha256_ctx_mgr_init_sse (SHA256_HASH_CTX_MGR *mgr)`  
SSE4.1

`sha256_ctx_mgr_submit (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)`  
SSE4.1 or AVX or AVX2

---

---

`sha256_ctx_mgr_submit_avx` (`SHA256_HASH_CTX_MGR *mgr`, `SHA256_HASH_CTX *ctx`, `const void *buffer`,  
`uint32_t len`, `HASH_CTX_FLAG flags`)  
AVX

`sha256_ctx_mgr_submit_avx2` (`SHA256_HASH_CTX_MGR *mgr`, `SHA256_HASH_CTX *ctx`, `const void *buffer`,  
`uint32_t len`, `HASH_CTX_FLAG flags`)  
AVX2

`sha256_ctx_mgr_submit_sse` (`SHA256_HASH_CTX_MGR *mgr`, `SHA256_HASH_CTX *ctx`, `const void *buffer`,  
`uint32_t len`, `HASH_CTX_FLAG flags`)  
SSE4.1

`sha256_flush_job` (`SHA256_MB_MGR *state`)  
SSE4.1

`sha256_flush_job_avx` (`SHA256_MB_MGR *state`)  
AVX

`sha256_flush_job_avx2` (`SHA256_MB_MGR_X8 *state`)  
AVX2

`sha256_init_mb_mgr` (`SHA256_MB_MGR *state`)  
SSE4.1

`sha256_init_mb_mgr_x8` (`SHA256_MB_MGR_X8 *state`)  
AVX2

`sha256_submit_job` (`SHA256_MB_MGR *state`, `JOB_SHA256 *job`)  
SSE4.1

`sha256_submit_job_avx` (`SHA256_MB_MGR *state`, `JOB_SHA256 *job`)  
AVX

`sha256_submit_job_avx2` (`SHA256_MB_MGR_X8 *state`, `JOB_SHA256 *job`)  
AVX2

`sha512_ctx_mgr_flush` (`SHA512_HASH_CTX_MGR *mgr`)  
SSE4.1 or AVX or AVX2

`sha512_ctx_mgr_flush_avx` (`SHA512_HASH_CTX_MGR *mgr`)  
AVX

`sha512_ctx_mgr_flush_avx2` (`SHA512_HASH_CTX_MGR *mgr`)  
AVX2

`sha512_ctx_mgr_flush_sb_sse4` (`SHA512_HASH_CTX_MGR *mgr`)  
SSE4

`sha512_ctx_mgr_flush_sse` (`SHA512_HASH_CTX_MGR *mgr`)  
SSE4.1

`sha512_ctx_mgr_init` (`SHA512_HASH_CTX_MGR *mgr`)  
SSE4.1 or AVX or AVX2

---

---

`sha512_ctx_mgr_init_avx` (SHA512\_HASH\_CTX\_MGR \*mgr)  
AVX

`sha512_ctx_mgr_init_avx2` (SHA512\_HASH\_CTX\_MGR \*mgr)  
AVX2

`sha512_ctx_mgr_init_sb_sse4` (SHA512\_HASH\_CTX\_MGR \*mgr)  
SSE4

`sha512_ctx_mgr_init_sse` (SHA512\_HASH\_CTX\_MGR \*mgr)  
SSE4.1

`sha512_ctx_mgr_submit` (SHA512\_HASH\_CTX\_MGR \*mgr, SHA512\_HASH\_CTX \*ctx, const void \*buffer,  
uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4.1 or AVX or AVX2

`sha512_ctx_mgr_submit_avx` (SHA512\_HASH\_CTX\_MGR \*mgr, SHA512\_HASH\_CTX \*ctx, const void \*buffer,  
uint32\_t len, HASH\_CTX\_FLAG flags)  
AVX

`sha512_ctx_mgr_submit_avx2` (SHA512\_HASH\_CTX\_MGR \*mgr, SHA512\_HASH\_CTX \*ctx, const void \*buffer,  
uint32\_t len, HASH\_CTX\_FLAG flags)  
AVX2

`sha512_ctx_mgr_submit_sb_sse4` (SHA512\_HASH\_CTX\_MGR \*mgr, SHA512\_HASH\_CTX \*ctx, const void  
\*buffer, uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4

`sha512_ctx_mgr_submit_sse` (SHA512\_HASH\_CTX\_MGR \*mgr, SHA512\_HASH\_CTX \*ctx, const void \*buffer,  
uint32\_t len, HASH\_CTX\_FLAG flags)  
SSE4.1

`sha512_flush_job` (SHA512\_MB\_MGR \*state)  
SSE4.1

`sha512_flush_job_avx` (SHA512\_MB\_MGR \*state)  
AVX

`sha512_flush_job_avx2` (SHA512\_MB\_MGR\_X4 \*state)  
AVX2

`sha512_init_mb_mgr` (SHA512\_MB\_MGR \*state)  
SSE4.1

`sha512_init_mb_mgr_x4` (SHA512\_MB\_MGR\_X4 \*state)  
AVX2

`sha512_submit_job` (SHA512\_MB\_MGR \*state, JOB\_SHA512 \*job)  
SSE4.1

`sha512_submit_job_avx` (SHA512\_MB\_MGR \*state, JOB\_SHA512 \*job)  
AVX

`sha512_submit_job_avx2` (SHA512\_MB\_MGR\_X4 \*state, JOB\_SHA512 \*job)  
AVX2

---

---

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

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

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

`XTS_AES_128_dec` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8 \*pt)  
AES-NI

`XTS_AES_128_dec_expanded_key` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8 \*pt)  
AES-NI

`XTS_AES_128_enc` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8 \*ct)  
AES-NI

`XTS_AES_128_enc_expanded_key` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8 \*ct)  
AES-NI

`XTS_AES_256_dec` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8 \*pt)  
AES-NI

`XTS_AES_256_dec_expanded_key` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8 \*pt)  
AES-NI

`XTS_AES_256_enc` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8 \*ct)  
AES-NI

`XTS_AES_256_enc_expanded_key` (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8 \*ct)  
AES-NI

---

## CHAPTER 4

# DATA STRUCTURE INDEX

---

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">BitBuf2</a>	Holds Bit Buffer information . . . . .	26
<a href="#">JOB_MD5</a>	Holds info describing a single MD5 job for the multi-buffer manager . . . . .	26
<a href="#">JOB_SHA1</a>	Holds info describing a single SHA1 job for the multi-buffer manager . . . . .	27
<a href="#">JOB_SHA256</a>	Holds info describing a single SHA256 job for the multi-buffer manager . . . . .	28
<a href="#">JOB_SHA512</a>	Holds info describing a single SHA512 job for the multi-buffer manager . . . . .	29
<a href="#">LZ_State1</a>	Holds the internal state information for input and output compression streams . . . . .	30
<a href="#">LZ_Stream1</a>	Holds stream information . . . . .	31
<a href="#">MD5_ARGS_X8</a>	Holds arguments for submitted MD5 job . . . . .	32
<a href="#">MD5_ARGS_X8X2</a>	Holds arguments for submitted AVX2 MD5 job . . . . .	33
<a href="#">MD5_HASH_CTX</a>	Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager . . . . .	33
<a href="#">MD5_HASH_CTX_MGR</a>	Context layer - Holds state for multi-buffer MD5 jobs . . . . .	34
<a href="#">MD5_HMAC_LANE_DATA</a>	MD5 out-of-order scheduler fields . . . . .	34
<a href="#">MD5_JOB</a>	Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager . . . . .	35
<a href="#">MD5_LANE_DATA</a>	Scheduler layer - Lane data . . . . .	35
<a href="#">MD5_MB_ARGS_X16</a>	Scheduler layer - Holds arguments for submitted MD5 job . . . . .	36
<a href="#">MD5_MB_JOB_MGR</a>	Scheduler layer - Holds state for multi-buffer MD5 jobs . . . . .	36
<a href="#">MD5_MB_MGR</a>	Holds state for multi-buffer MD5 jobs . . . . .	37
<a href="#">MD5_MB_MGR_X8X2</a>	Holds state for multi-buffer AVX2 MD5 jobs . . . . .	37



<a href="#">SHA1_ARGS_X4</a>	
Holds arguments for submitted SHA1 job . . . . .	38
<a href="#">SHA1_ARGS_X8</a>	
Holds arguments for submitted SHA1 job . . . . .	38
<a href="#">SHA1_HASH_CTX</a>	
Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager . . . .	39
<a href="#">SHA1_HASH_CTX_MGR</a>	
Context layer - Holds state for multi-buffer SHA1 jobs . . . . .	40
<a href="#">SHA1_HMAC_LANE_DATA</a>	
SHA1 out-of-order scheduler fields . . . . .	40
<a href="#">SHA1_JOB</a>	
Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager . . . . .	41
<a href="#">SHA1_LANE_DATA</a>	
Scheduler layer - Lane data . . . . .	41
<a href="#">SHA1_MB_ARGS_X8</a>	
Scheduler layer - Holds arguments for submitted SHA1 job . . . . .	42
<a href="#">SHA1_MB_JOB_MGR</a>	
Scheduler layer - Holds state for multi-buffer SHA1 jobs . . . . .	42
<a href="#">SHA1_MB_MGR</a>	
Holds state for multi-buffer SHA1 jobs . . . . .	42
<a href="#">SHA1_MB_MGR_X8</a>	
Holds state for multi-buffer SHA1 jobs . . . . .	43
<a href="#">SHA256_ARGS_X4</a>	
Holds arguments for submitted SHA256 job . . . . .	44
<a href="#">SHA256_ARGS_X8</a>	
Holds arguments for submitted SHA256 job . . . . .	44
<a href="#">SHA256_HASH_CTX</a>	
Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager . .	45
<a href="#">SHA256_HASH_CTX_MGR</a>	
Context layer - Holds state for multi-buffer SHA256 jobs . . . . .	46
<a href="#">SHA256_HMAC_LANE_DATA</a>	
SHA256 out-of-order scheduler fields . . . . .	46
<a href="#">SHA256_JOB</a>	
Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager . . . .	47
<a href="#">SHA256_LANE_DATA</a>	
Scheduler layer - Lane data . . . . .	47
<a href="#">SHA256_MB_ARGS_X8</a>	
Scheduler layer - Holds arguments for submitted SHA256 job . . . . .	48
<a href="#">SHA256_MB_JOB_MGR</a>	
Scheduler layer - Holds state for multi-buffer SHA256 jobs . . . . .	48
<a href="#">SHA256_MB_MGR</a>	
Holds state for multi-buffer SHA256 jobs . . . . .	48
<a href="#">SHA256_MB_MGR_X8</a>	
Holds state for multi-buffer SHA256 jobs . . . . .	49
<a href="#">SHA512_ARGS_X2</a>	
Holds arguments for submitted SHA512 job . . . . .	50

<a href="#">SHA512_ARGS_X4</a>	
Holds arguments for submitted AVX2 SHA512 job . . . . .	50
<a href="#">SHA512_HASH_CTX</a>	
Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager . .	51
<a href="#">SHA512_HASH_CTX_MGR</a>	
Context layer - Holds state for multi-buffer SHA512 jobs . . . . .	51
<a href="#">SHA512_HMAC_LANE_DATA</a>	
SHA512 out-of-order scheduler fields . . . . .	52
<a href="#">SHA512_JOB</a>	
Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager . . . .	52
<a href="#">SHA512_LANE_DATA</a>	
Scheduler layer - Lane data . . . . .	53
<a href="#">SHA512_MB_ARGS_X4</a>	
Scheduler layer - Holds arguments for submitted SHA512 job . . . . .	53
<a href="#">SHA512_MB_JOB_MGR</a>	
Scheduler layer - Holds state for multi-buffer SHA512 jobs . . . . .	54
<a href="#">SHA512_MB_MGR</a>	
Holds state for multi-buffer SHA512 jobs . . . . .	54
<a href="#">SHA512_MB_MGR_X4</a>	
Holds state for multi-buffer SHA512 jobs . . . . .	55

---

### 5.1 File List

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

<a href="#">aes_xts.h</a>	AES XTS encryption function prototypes . . . . .	56
<a href="#">crc.h</a>	CRC functions . . . . .	62
<a href="#">erasure_code.h</a>	Interface to functions supporting erasure code encode and decode . . . . .	68
<a href="#">gf_vect_mul.h</a>	Interface to functions for vector (block) multiplication in $GF(2^8)$ . . . . .	94
<a href="#">igzip_lib.h</a>	This file defines the igzip compression interface, a high performance deflate compression interface for storage applications . . . . .	97
<a href="#">intrinreg.h</a>	Defines intrinsic types used by the new hashing API . . . . .	101
<a href="#">mb_md5.h</a>	Multi-buffer MD5 function prototypes and structures to submit jobs . . . . .	101
<a href="#">mb_sha1.h</a>	Multi-buffer SHA1 function prototypes and structures to submit jobs . . . . .	106
<a href="#">mb_sha256.h</a>	Multi-buffer SHA256 function prototypes and structures to submit jobs . . . . .	110
<a href="#">mb_sha512.h</a>	Multi-buffer SHA512 function prototypes and structures to submit jobs . . . . .	115
<a href="#">md5_mb.h</a>	Multi-buffer CTX API MD5 function prototypes and structures . . . . .	119
<a href="#">mem_routines.h</a>	Interface to storage mem operations . . . . .	126
<a href="#">memcpy_inline.h</a>	Defines intrinsic memcpy functions used by the new hashing API . . . . .	129
<a href="#">multi_buffer.h</a>	Multi-buffer common fields . . . . .	129
<a href="#">raid.h</a>	Interface to RAID functions - XOR and P+Q calculation . . . . .	131
<a href="#">sha.h</a>	SHA1 functions . . . . .	138
<a href="#">sha1_mb.h</a>	Multi-buffer CTX API SHA1 function prototypes and structures . . . . .	139
<a href="#">sha256_mb.h</a>	Multi-buffer CTX API SHA256 function prototypes and structures . . . . .	147

---

<a href="#">sha512_mb.h</a>	Single/Multi-buffer CTX API SHA512 function prototypes and structures . . . . .	154
<a href="#">types.h</a>	Defines standard width types . . . . .	163

---

## CHAPTER 6

# DATA STRUCTURE DOCUMENTATION

---

### 6.1 BitBuf2 Struct Reference

Holds Bit Buffer information.

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

#### Data Fields

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

#### 6.1.1 Detailed Description

Holds Bit Buffer information.

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

- [igzip\\_lib.h](#)

### 6.2 JOB\_MD5 Struct Reference

Holds info describing a single MD5 job for the multi-buffer manager.

```
#include <mb_md5.h>
```

#### Data Fields

- `UINT8 * buffer`

- pointer to data buffer for this job*
- `UINT32 len`
  - length of buffer for this job in bytes. For finalize can be any length. For update must be a multiple of MD5\_BLOCK\_SIZE.*
- `UINT32 len_total`
  - total size of the complete hash in bytes*
- `ALIGN UINT32 result_digest [4]`
  - holds result of hash operation*
- `JOB_STS status`
  - output job status*
- `UINT32 flags`
  - input flags to indicate init, update or finalize*
- `void * user_data`
  - pointer for user to keep any job-related data*

### 6.2.1 Detailed Description

Holds info describing a single MD5 job for the multi-buffer manager.

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

- `mb_md5.h`

## 6.3 JOB\_SHA1 Struct Reference

Holds info describing a single SHA1 job for the multi-buffer manager.

```
#include <mb_sha1.h>
```

### Data Fields

- `UINT8 * buffer`
    - pointer to data buffer for this job*
  - `UINT32 len`
    - length of buffer for this job in bytes. For finalize can be any length. For update must be a multiple of SHA1\_BLOCK\_SIZE.*
  - `UINT32 len_total`
    - total size of the complete hash in bytes*
  - `ALIGN UINT32 result_digest [5]`
    - holds result of hash operation*
  - `JOB_STS status`
    - output job status*
-

- `UINT32 flags`  
*input flags to indicate init, update or finalize*
- `void * user_data`  
*pointer for user to keep any job-related data*

### 6.3.1 Detailed Description

Holds info describing a single SHA1 job for the multi-buffer manager.

Examples:

[multi\\_buffer\\_sha1\\_example.c](#).

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

- [mb\\_sha1.h](#)

## 6.4 JOB\_SHA256 Struct Reference

Holds info describing a single SHA256 job for the multi-buffer manager.

```
#include <mb_sha256.h>
```

### Data Fields

- `UINT8 * buffer`  
*pointer to data buffer for this job*
  - `UINT32 len`  
*length of buffer for this job in bytes. For finalize can be any length. For update must be a multiple of SHA256\_BLOCK\_SIZE.*
  - `UINT32 len_total`  
*total size of the complete hash in bytes*
  - `ALIGN UINT32 result_digest [8]`  
*holds result of hash operation*
  - `JOB_STS status`  
*output job status*
  - `UINT32 flags`  
*input flags to indicate init, update or finalize*
  - `void * user_data`  
*pointer for user to keep any job-related data*
-

### 6.4.1 Detailed Description

Holds info describing a single SHA256 job for the multi-buffer manager.

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

- [mb\\_sha256.h](#)

## 6.5 JOB\_SHA512 Struct Reference

Holds info describing a single SHA512 job for the multi-buffer manager.

```
#include <mb_sha512.h>
```

### Data Fields

- `UINT8 * buffer`  
*pointer to data buffer for this job*
- `UINT32 len`  
*length of buffer for this job in bytes. For finalize can be any length. For update must be a multiple of SHA512\_BLOCK\_SIZE.*
- `UINT32 len\_total`  
*total size of the complete hash in bytes*
- `ALIGN_UINT64 result\_digest [8]`  
*holds result of hash operation*
- `JOB_STS status`  
*output job status*
- `UINT32 flags`  
*input flags to indicate init, update or finalize*
- `void * user\_data`  
*pointer for user to keep any job-related data*

### 6.5.1 Detailed Description

Holds info describing a single SHA512 job for the multi-buffer manager.

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

- [mb\\_sha512.h](#)
-



## 6.6 LZ\_State1 Struct Reference

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

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

### Data Fields

- uint32\_t [b\\_bytes\\_valid](#)  
*number of bytes of valid data in buffer*
  - uint32\_t [b\\_bytes\\_processed](#)  
*keeps track of the number of bytes processed from the input buffer*
  - uint8\_t \* [file\\_start](#)  
*pointer to where file would logically start*
  - ALIGN uint32\_t [crc](#) [16]  
*actually 4 128-bit integers*
  - struct [BitBuf2](#) [bitbuf](#)  
*Bit Buffer.*
  - enum [LZ\\_State1\\_state](#) [state](#)  
*can be LZS2\_HDR, LZS2\_BODY, LZS2\_TRL, LZS2\_END*
  - 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 [last\\_flush](#)  
*keeps track of last submitted flush*
  - uint32\_t [submitted](#)  
*keeps track of submitted bytes internally*
  - uint8\_t \* [last\\_next\\_in](#)  
*keeps track of last submitted input buffer*
  - uint32\_t [has\\_eob](#)  
*keeps track of eob on the last deflate block*
  - uint32\_t [has\\_eob\\_hdr](#)  
*keeps track of eob hdr (with BFINAL set)*
  - uint32\_t [stored\\_blk\\_len](#)  
*keeps track of the length of a stored block*
  - uint32\_t [no\\_comp](#)
-

*used to copy data into history where a stored block is used*

- `uint32_t left_over`  
*keeps track of overflow bytes*
- `uint32_t overflow_submitted`  
*keeps track of how many bytes were submitted when overflow*
- `uint32_t overflow`  
*indicates we're in an overflow state*
- `uint32_t had_overflow`  
*indicates we had overflow state*
- `ALIGN uint8_t buffer [(2*(8*1024)+(17*16))+16]`  
*Internal buffer.*
- `ALIGN uint16_t head [(8*1024)]`  
*Hash array.*

### 6.6.1 Detailed Description

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

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

- [igzip\\_lib.h](#)

## 6.7 LZ\_Stream1 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*
- uint32\_t [end\\_of\\_stream](#)  
*non-zero if this is the last input buffer*
- uint32\_t [flush](#)  
*Flush type can be FINISH\_FLUSH or SYNC\_FLUSH.*
- uint32\_t [bytes\\_consumed](#)  
*indicates the number of bytes processed from the input buffer*
- struct [LZ\\_State1](#) [internal\\_state](#)  
*Internal state for this stream.*

### 6.7.1 Detailed Description

Holds stream information.

Examples:

[igzip\\_example.c](#).

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

- [igzip\\_lib.h](#)

## 6.8 MD5\_ARGS\_X8 Struct Reference

Holds arguments for submitted MD5 job.

```
#include <mb_md5.h>
```

### Data Fields

- ALIGN\_UINT32 [digest](#) [4][8]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_MD5\_LANES]  
*Pointers to working buffer for each lane.*

### 6.8.1 Detailed Description

Holds arguments for submitted MD5 job.

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

- [mb\\_md5.h](#)
-

## 6.9 MD5\_ARGS\_X8X2 Struct Reference

Holds arguments for submitted AVX2 MD5 job.

```
#include <mb_md5.h>
```

### Data Fields

- ALIGN UINT32 [digest](#) [4][16]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_MD5\_LANES\_X8X2]  
*Pointers to working buffer for each lane.*

### 6.9.1 Detailed Description

Holds arguments for submitted AVX2 MD5 job.

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

- [mb\\_md5.h](#)

## 6.10 MD5\_HASH\_CTX Struct Reference

Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.

```
#include <md5_mb.h>
```

### Data Fields

- [HASH\\_CTX\\_STS](#) [status](#)  
*Context status flag.*
  - [HASH\\_CTX\\_ERROR](#) [error](#)  
*Context error flag.*
  - uint32\_t [total\\_length](#)  
*Running counter of length processed for this CTX's job.*
  - const void \* [incoming\\_buffer](#)  
*pointer to data input buffer for this CTX's job*
  - uint32\_t [incoming\\_buffer\\_length](#)  
*length of buffer for this job in bytes.*
  - uint8\_t [partial\\_block\\_buffer](#) [MD5\_BLOCK\_SIZE \*2]  
*CTX partial blocks.*
-

- void \* [user\\_data](#)  
*pointer for user to keep any job-related data*

### 6.10.1 Detailed Description

Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.

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

- [md5\\_mb.h](#)

## 6.11 MD5\_HASH\_CTX\_MGR Struct Reference

Context layer - Holds state for multi-buffer MD5 jobs.

```
#include <md5_mb.h>
```

### 6.11.1 Detailed Description

Context layer - Holds state for multi-buffer MD5 jobs.

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

- [md5\\_mb.h](#)

## 6.12 MD5\_HMAC\_LANE\_DATA Struct Reference

MD5 out-of-order scheduler fields.

```
#include <mb_md5.h>
```

### Data Fields

- ALIGN\_UINT8 [extra\\_block](#) [2 \* 64 + 8]  
*Extra block array - for padding or sub-block message.*
  - JOB\_MD5 \* [job\\_in\\_lane](#)  
*address of lane's current job*
  - UINT32 [extra\\_blocks](#)  
*num extra blocks (1 or 2)*
  - UINT32 [size\\_offset](#)  
*offset in extra\_block to start of size field*
-

- `UINT32 start_offset`  
*offset to start of data*

### 6.12.1 Detailed Description

MD5 out-of-order scheduler fields.

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

- [mb\\_md5.h](#)

## 6.13 MD5\_JOB Struct Reference

Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.

```
#include <md5_mb.h>
```

### Data Fields

- `uint8_t * buffer`  
*pointer to data buffer for this job*
- `uint32_t len`  
*length of buffer for this job in blocks.*
- `JOB_STS status`  
*output job status*
- `void * user_data`  
*pointer for user's job-related data*

### 6.13.1 Detailed Description

Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.

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

- [md5\\_mb.h](#)

## 6.14 MD5\_LANE\_DATA Struct Reference

Scheduler layer - Lane data.

```
#include <md5_mb.h>
```

---

### 6.14.1 Detailed Description

Scheduler layer - Lane data.

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

- [md5\\_mb.h](#)

## 6.15 MD5\_MB\_ARGS\_X16 Struct Reference

Scheduler layer - Holds arguments for submitted MD5 job.

```
#include <md5_mb.h>
```

### 6.15.1 Detailed Description

Scheduler layer - Holds arguments for submitted MD5 job.

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

- [md5\\_mb.h](#)

## 6.16 MD5\_MB\_JOB\_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer MD5 jobs.

```
#include <md5_mb.h>
```

### Data Fields

- `uint64_t unused_lanes`  
*each nibble is index (0...3 or 0...7) of unused lanes, nibble 4 or 8 is set to F as a flag*

### 6.16.1 Detailed Description

Scheduler layer - Holds state for multi-buffer MD5 jobs.

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

- [md5\\_mb.h](#)
-

## 6.17 MD5\_MB\_MGR Struct Reference

Holds state for multi-buffer MD5 jobs.

```
#include <mb_md5.h>
```

### Data Fields

- [MD5\\_ARGS\\_X8 args](#)  
*Structure containing working digests and pointers to input buffers.*
- ALIGN UINT32 [lens](#) [8]  
*Length (number of blocks) of each lane's current message.*
- UINT64 [unused\\_lanes](#)  
*each nibble is index (0...7) of unused lanes, nibble 8 is set to F as a flag*
- [MD5\\_HMAC\\_LANE\\_DATA ldata](#) [NUM\_MD5\_LANES]  
*Structure containing lane setup.*

### 6.17.1 Detailed Description

Holds state for multi-buffer MD5 jobs.

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

- [mb\\_md5.h](#)

## 6.18 MD5\_MB\_MGR\_X8X2 Struct Reference

Holds state for multi-buffer AVX2 MD5 jobs.

```
#include <mb_md5.h>
```

### Data Fields

- [MD5\\_ARGS\\_X8X2 args](#)  
*Structure containing working digests and pointers to input buffers.*
  - ALIGN UINT32 [lens](#) [16]  
*Length (number of blocks) of each lane's current message.*
  - UINT64 [unused\\_lanes](#)  
*each nibble is index (0...15) of unused lanes*
  - [MD5\\_HMAC\\_LANE\\_DATA ldata](#) [NUM\_MD5\_LANES\_X8X2]  
*Structure containing lane setup.*
-



- UINT32 [num\\_lanes\\_inuse](#)

*Counter required to supplement unused\_lanes in the case of 16 lanes.*

### 6.18.1 Detailed Description

Holds state for multi-buffer AVX2 MD5 jobs.

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

- [mb\\_md5.h](#)

## 6.19 SHA1\_ARGS\_X4 Struct Reference

Holds arguments for submitted SHA1 job.

```
#include <mb_sha1.h>
```

### Data Fields

- ALIGN UINT32 [digest](#) [5][4]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_SHA1\_LANES]  
*Pointers to working buffer for each lane.*

### 6.19.1 Detailed Description

Holds arguments for submitted SHA1 job.

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

- [mb\\_sha1.h](#)

## 6.20 SHA1\_ARGS\_X8 Struct Reference

Holds arguments for submitted SHA1 job.

```
#include <mb_sha1.h>
```

### Data Fields

- ALIGN UINT32 [digest](#) [5][8]
-

*Holds the working digest for each lane.*

- `UINT8 * data_ptr [NUM_SHA1_LANES_X8]`

*Pointers to working buffer for each lane.*

### 6.20.1 Detailed Description

Holds arguments for submitted SHA1 job.

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

- [mb\\_sha1.h](#)

## 6.21 SHA1\_HASH\_CTX Struct Reference

Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.

```
#include <sha1_mb.h>
```

### Data Fields

- [HASH\\_CTX\\_STS](#) status  
*Context status flag.*
- [HASH\\_CTX\\_ERROR](#) error  
*Context error flag.*
- `uint32_t total_length`  
*Running counter of length processed for this CTX's job.*
- `const void * incoming_buffer`  
*pointer to data input buffer for this CTX's job*
- `uint32_t incoming_buffer_length`  
*length of buffer for this job in bytes.*
- `uint8_t partial_block_buffer [SHA1_BLOCK_SIZE *2]`  
*CTX partial blocks.*
- `void * user_data`  
*pointer for user to keep any job-related data*

### 6.21.1 Detailed Description

Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.

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

- [sha1\\_mb.h](#)
-

## 6.22 SHA1\_HASH\_CTX\_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA1 jobs.

```
#include <sha1_mb.h>
```

### 6.22.1 Detailed Description

Context layer - Holds state for multi-buffer SHA1 jobs.

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

- [sha1\\_mb.h](#)

## 6.23 SHA1\_HMAC\_LANE\_DATA Struct Reference

SHA1 out-of-order scheduler fields.

```
#include <mb_sha1.h>
```

### Data Fields

- ALIGN\_UINT8 [extra\\_block](#) [2 \* 64 + 8]  
*Extra block array - for padding or sub-block message.*
- JOB\_SHA1 \* [job\\_in\\_lane](#)  
*address of lane's current job*
- UINT32 [extra\\_blocks](#)  
*num extra blocks (1 or 2)*
- UINT32 [size\\_offset](#)  
*offset in extra\_block to start of size field*
- UINT32 [start\\_offset](#)  
*offset to start of data*
- UINT32 [padding](#)  
*padding for internal use*

### 6.23.1 Detailed Description

SHA1 out-of-order scheduler fields.

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

- [mb\\_sha1.h](#)
-

## 6.24 SHA1\_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.

```
#include <sha1_mb.h>
```

### Data Fields

- `uint8_t * buffer`  
*pointer to data buffer for this job*
- `uint32_t len`  
*length of buffer for this job in blocks.*
- `JOB_STS status`  
*output job status*
- `void * user_data`  
*pointer for user's job-related data*

#### 6.24.1 Detailed Description

Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.

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

- [sha1\\_mb.h](#)

## 6.25 SHA1\_LANE\_DATA Struct Reference

Scheduler layer - Lane data.

```
#include <sha1_mb.h>
```

#### 6.25.1 Detailed Description

Scheduler layer - Lane data.

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

- [sha1\\_mb.h](#)
-

## 6.26 SHA1\_MB\_ARGS\_X8 Struct Reference

Scheduler layer - Holds arguments for submitted SHA1 job.

```
#include <sha1_mb.h>
```

### 6.26.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA1 job.

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

- [sha1\\_mb.h](#)

## 6.27 SHA1\_MB\_JOB\_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA1 jobs.

```
#include <sha1_mb.h>
```

### Data Fields

- `uint64_t unused_lanes`  
*each nibble is index (0...3 or 0...7) of unused lanes, nibble 4 or 8 is set to F as a flag*

### 6.27.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA1 jobs.

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

- [sha1\\_mb.h](#)

## 6.28 SHA1\_MB\_MGR Struct Reference

Holds state for multi-buffer SHA1 jobs.

```
#include <mb_sha1.h>
```

### Data Fields

- [SHA1\\_ARGS\\_X4 args](#)
-

*Structure containing working digests and pointers to input buffers.*

- `UINT64 lens` [NUM\_SHA1\_LANES]

*Length (number of blocks) of each lane's current message.*

- `UINT64 unused_lanes`

*each byte is index (0...3) of unused lanes, byte 4 is set to FF as a flag*

- `SHA1_HMAC_LANE_DATA ldata` [NUM\_SHA1\_LANES]

*Structure containing lane setup.*

### 6.28.1 Detailed Description

Holds state for multi-buffer SHA1 jobs.

Examples:

[multi\\_buffer\\_sha1\\_example.c](#).

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

- [mb\\_sha1.h](#)

## 6.29 SHA1\_MB\_MGR\_X8 Struct Reference

Holds state for multi-buffer SHA1 jobs.

```
#include <mb_sha1.h>
```

### Data Fields

- `SHA1_ARGS_X8 args`

*Structure containing working digests and pointers to input buffers.*

- `ALIGN UINT32 lens` [8]

*Length (number of blocks) of each lane's current message.*

- `UINT64 unused_lanes`

*each nibble is index (0...7) of unused lanes, nibble 8 is set to F as a flag*

- `SHA1_HMAC_LANE_DATA ldata` [NUM\_SHA1\_LANES\_X8]

*Structure containing lane setup.*

---

### 6.29.1 Detailed Description

Holds state for multi-buffer SHA1 jobs.

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

- [mb\\_sha1.h](#)

## 6.30 SHA256\_ARGS\_X4 Struct Reference

Holds arguments for submitted SHA256 job.

```
#include <mb_sha256.h>
```

### Data Fields

- ALIGN UINT32 [digest](#) [8][4]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_SHA256\_LANES]  
*Pointers to working buffer for each lane.*

### 6.30.1 Detailed Description

Holds arguments for submitted SHA256 job.

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

- [mb\\_sha256.h](#)

## 6.31 SHA256\_ARGS\_X8 Struct Reference

Holds arguments for submitted SHA256 job.

```
#include <mb_sha256.h>
```

### Data Fields

- ALIGN UINT32 [digest](#) [8][8]  
*Holds the working digest for each lane.*
  - UINT8 \* [data\\_ptr](#) [NUM\_SHA256\_LANES\_X8]  
*Pointers to working buffer for each lane.*
-

### 6.31.1 Detailed Description

Holds arguments for submitted SHA256 job.

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

- [mb\\_sha256.h](#)

## 6.32 SHA256\_HASH\_CTX Struct Reference

Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.

```
#include <sha256_mb.h>
```

### Data Fields

- [HASH\\_CTX\\_STS](#) status  
*Context status flag.*
- [HASH\\_CTX\\_ERROR](#) error  
*Context error flag.*
- `uint32_t` [total\\_length](#)  
*Running counter of length processed for this CTX's job.*
- `const void *` [incoming\\_buffer](#)  
*pointer to data input buffer for this CTX's job*
- `uint32_t` [incoming\\_buffer\\_length](#)  
*length of buffer for this job in bytes.*
- `uint8_t` [partial\\_block\\_buffer](#) [SHA256\_BLOCK\_SIZE \*2]  
*CTX partial blocks.*
- `void *` [user\\_data](#)  
*pointer for user to keep any job-related data*

### 6.32.1 Detailed Description

Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.

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

- [sha256\\_mb.h](#)
-



## 6.33 SHA256\_HASH\_CTX\_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA256 jobs.

```
#include <sha256_mb.h>
```

### 6.33.1 Detailed Description

Context layer - Holds state for multi-buffer SHA256 jobs.

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

- [sha256\\_mb.h](#)

## 6.34 SHA256\_HMAC\_LANE\_DATA Struct Reference

SHA256 out-of-order scheduler fields.

```
#include <mb_sha256.h>
```

### Data Fields

- ALIGN\_UINT8 [extra\\_block](#) [2 \* 64 + 8]  
*Extra block array - for padding or sub-block message.*
- [JOB\\_SHA256](#) \* [job\\_in\\_lane](#)  
*address of lane's current job*
- UINT32 [extra\\_blocks](#)  
*num extra blocks (1 or 2)*
- UINT32 [size\\_offset](#)  
*offset in extra\_block to start of size field*
- UINT32 [start\\_offset](#)  
*offset to start of data*
- UINT32 [padding](#)  
*padding for internal use*

### 6.34.1 Detailed Description

SHA256 out-of-order scheduler fields.

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

- [mb\\_sha256.h](#)
-

## 6.35 SHA256\_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.

```
#include <sha256_mb.h>
```

### Data Fields

- `uint8_t * buffer`  
*pointer to data buffer for this job*
- `uint64_t len`  
*length of buffer for this job in blocks.*
- `JOB_STS status`  
*output job status*
- `void * user_data`  
*pointer for user's job-related data*

### 6.35.1 Detailed Description

Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.

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

- [sha256\\_mb.h](#)

## 6.36 SHA256\_LANE\_DATA Struct Reference

Scheduler layer - Lane data.

```
#include <sha256_mb.h>
```

### 6.36.1 Detailed Description

Scheduler layer - Lane data.

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

- [sha256\\_mb.h](#)
-

## 6.37 SHA256\_MB\_ARGS\_X8 Struct Reference

Scheduler layer - Holds arguments for submitted SHA256 job.

```
#include <sha256_mb.h>
```

### 6.37.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA256 job.

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

- [sha256\\_mb.h](#)

## 6.38 SHA256\_MB\_JOB\_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA256 jobs.

```
#include <sha256_mb.h>
```

### Data Fields

- `uint64_t unused_lanes`  
*each nibble is index (0...3 or 0...7) of unused lanes, nibble 4 or 8 is set to F as a flag*

### 6.38.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA256 jobs.

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

- [sha256\\_mb.h](#)

## 6.39 SHA256\_MB\_MGR Struct Reference

Holds state for multi-buffer SHA256 jobs.

```
#include <mb_sha256.h>
```

### Data Fields

- [SHA256\\_ARGS\\_X4 args](#)
-

*Structure containing working digests and pointers to input buffers.*

- UINT64 [lens](#) [NUM\_SHA256\_LANES]

*Length (number of blocks) of each lane's current message.*

- UINT64 [unused\\_lanes](#)

*each byte is index (0...3) of unused lanes, byte 4 is set to FF as a flag*

- [SHA256\\_HMAC\\_LANE\\_DATA](#) *ldata* [NUM\_SHA256\_LANES]

*Structure containing lane setup.*

### 6.39.1 Detailed Description

Holds state for multi-buffer SHA256 jobs.

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

- [mb\\_sha256.h](#)

## 6.40 SHA256\_MB\_MGR\_X8 Struct Reference

Holds state for multi-buffer SHA256 jobs.

```
#include <mb_sha256.h>
```

### Data Fields

- [SHA256\\_ARGS\\_X8](#) *args*

*Structure containing working digests and pointers to input buffers.*

- ALIGN UINT32 [lens](#) [8]

*Length (number of blocks) of each lane's current message.*

- UINT64 [unused\\_lanes](#)

*each nibble is index (0...7) of unused lanes nibble 8 is set to F as a flag*

- [SHA256\\_HMAC\\_LANE\\_DATA](#) *ldata* [NUM\_SHA256\_LANES\_X8]

*Structure containing lane setup.*

### 6.40.1 Detailed Description

Holds state for multi-buffer SHA256 jobs.

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

- [mb\\_sha256.h](#)
-

## 6.41 SHA512\_ARGS\_X2 Struct Reference

Holds arguments for submitted SHA512 job.

```
#include <mb_sha512.h>
```

### Data Fields

- ALIGN UINT64 [digest](#) [8][2]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_SHA512\_LANES]  
*Pointers to working buffer for each lane.*

### 6.41.1 Detailed Description

Holds arguments for submitted SHA512 job.

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

- [mb\\_sha512.h](#)

## 6.42 SHA512\_ARGS\_X4 Struct Reference

Holds arguments for submitted AVX2 SHA512 job.

```
#include <mb_sha512.h>
```

### Data Fields

- ALIGN UINT64 [digest](#) [8][4]  
*Holds the working digest for each lane.*
- UINT8 \* [data\\_ptr](#) [NUM\_SHA512\_LANES\_X4]  
*Pointers to working buffer for each lane.*

### 6.42.1 Detailed Description

Holds arguments for submitted AVX2 SHA512 job.

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

- [mb\\_sha512.h](#)
-

## 6.43 SHA512\_HASH\_CTX Struct Reference

Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.

```
#include <sha512_mb.h>
```

### Data Fields

- [HASH\\_CTX\\_STS](#) *status*  
*Context status flag.*
- [HASH\\_CTX\\_ERROR](#) *error*  
*Context error flag.*
- [uint32\\_t](#) *total\_length*  
*Running counter of length processed for this CTX's job.*
- `const void *` [incoming\\_buffer](#)  
*pointer to data input buffer for this CTX's job*
- [uint32\\_t](#) *incoming\_buffer\_length*  
*length of buffer for this job in bytes.*
- [uint8\\_t](#) *partial\_block\_buffer* [SHA512\_BLOCK\_SIZE \*2]  
*CTX partial blocks.*
- `void *` [user\\_data](#)  
*pointer for user to keep any job-related data*

### 6.43.1 Detailed Description

Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.

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

- [sha512\\_mb.h](#)

## 6.44 SHA512\_HASH\_CTX\_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA512 jobs.

```
#include <sha512_mb.h>
```

### 6.44.1 Detailed Description

Context layer - Holds state for multi-buffer SHA512 jobs.

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

---

- [sha512\\_mb.h](#)

## 6.45 SHA512\_HMAC\_LANE\_DATA Struct Reference

SHA512 out-of-order scheduler fields.

```
#include <mb_sha512.h>
```

### Data Fields

- ALIGN\_UINT8 [extra\\_block](#) [2 \* 128 + 16]  
*extra block array - for padding or sub-block message*
- [JOB\\_SHA512](#) \* [job\\_in\\_lane](#)  
*address of lane's current job*
- UINT32 [extra\\_blocks](#)  
*num extra blocks (1 or 2)*
- UINT32 [size\\_offset](#)  
*offset in extra\_block to start of size field*
- UINT32 [start\\_offset](#)  
*offset to start of data*
- UINT32 [padding](#)  
*padding for internal use*

### 6.45.1 Detailed Description

SHA512 out-of-order scheduler fields.

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

- [mb\\_sha512.h](#)

## 6.46 SHA512\_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.

```
#include <sha512_mb.h>
```

---

## Data Fields

- `uint8_t * buffer`  
*pointer to data buffer for this job*
- `uint64_t len`  
*length of buffer for this job in blocks.*
- `JOB_STS status`  
*output job status*
- `void * user_data`  
*pointer for user's job-related data*

### 6.46.1 Detailed Description

Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.

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

- [sha512\\_mb.h](#)

## 6.47 SHA512\_LANE\_DATA Struct Reference

Scheduler layer - Lane data.

```
#include <sha512_mb.h>
```

### 6.47.1 Detailed Description

Scheduler layer - Lane data.

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

- [sha512\\_mb.h](#)

## 6.48 SHA512\_MB\_ARGS\_X4 Struct Reference

Scheduler layer - Holds arguments for submitted SHA512 job.

```
#include <sha512_mb.h>
```

---



### 6.48.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA512 job.

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

- [sha512\\_mb.h](#)

## 6.49 SHA512\_MB\_JOB\_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA512 jobs.

```
#include <sha512_mb.h>
```

### Data Fields

- `uint64_t unused_lanes`  
*each byte is index (00, 01 or 00...03) of unused lanes, byte 2 or 4 is set to FF as a flag*

### 6.49.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA512 jobs.

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

- [sha512\\_mb.h](#)

## 6.50 SHA512\_MB\_MGR Struct Reference

Holds state for multi-buffer SHA512 jobs.

```
#include <mb_sha512.h>
```

### Data Fields

- [SHA512\\_ARGS\\_X2 args](#)  
*Structure containing working digests and pointers to input buffers.*
  - `UINT64 lens [NUM_SHA512_LANES]`  
*Length (number of blocks) of each lane's current message.*
  - `UINT64 unused_lanes`  
*each byte is index (0...1) of unused lanes, byte 2 is set to FF as a flag*
-

- [SHA512\\_HMAC\\_LANE\\_DATA ldata](#) [NUM\_SHA512\_LANES]

*Structure containing lane setup.*

### 6.50.1 Detailed Description

Holds state for multi-buffer SHA512 jobs.

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

- [mb\\_sha512.h](#)

## 6.51 SHA512\_MB\_MGR\_X4 Struct Reference

Holds state for multi-buffer SHA512 jobs.

```
#include <mb_sha512.h>
```

### Data Fields

- [SHA512\\_ARGS\\_X4 args](#)  
*Structure containing working digests and pointers to input buffers.*
- ALIGN UINT32 [lens](#) [4]  
*Length (number of blocks) of each lane's current message.*
- UINT64 [unused\\_lanes](#)  
*each byte is index (0...1) of unused lanes, byte 4 is set to FF as a flag*
- [SHA512\\_HMAC\\_LANE\\_DATA ldata](#) [NUM\_SHA512\_LANES\_X4]  
*Structure containing lane setup.*

### 6.51.1 Detailed Description

Holds state for multi-buffer SHA512 jobs.

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

- [mb\\_sha512.h](#)
-

### 7.1 aes\_xts.h File Reference

AES XTS encryption function prototypes.

```
#include "types.h"
```

#### Functions

- void [aes\\_keyexp\\_128](#) (UINT8 \*key, UINT8 \*exp\_key\_enc, UINT8 \*exp\_key\_dec)  
*AES-128 key expansion for encryption and decryption.*
- void [XTS\\_AES\\_128\\_enc](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8 \*ct)  
*XTS-AES-128 Encryption.*
- void [XTS\\_AES\\_128\\_enc\\_expanded\\_key](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UI-  
NT8 \*pt, UINT8 \*ct)  
*XTS-AES-128 Encryption with pre-expanded keys.*
- void [XTS\\_AES\\_128\\_dec](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8  
\*pt)  
*XTS-AES-128 Decryption.*
- void [XTS\\_AES\\_128\\_dec\\_expanded\\_key](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UI-  
NT8 \*ct, UINT8 \*pt)  
*XTS-AES-128 Decryption with pre-expanded keys.*
- void [aes\\_keyexp\\_256](#) (UINT8 \*key, UINT8 \*exp\_key\_enc, UINT8 \*exp\_key\_dec)  
*AES-256 key expansion for encryption and decryption.*
- void [XTS\\_AES\\_256\\_enc](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*pt, UINT8  
\*ct)  
*XTS-AES-256 Encryption.*
- void [XTS\\_AES\\_256\\_enc\\_expanded\\_key](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UI-  
NT8 \*pt, UINT8 \*ct)  
*XTS-AES-256 Encryption with pre-expanded keys.*
- void [XTS\\_AES\\_256\\_dec](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UINT8 \*ct, UINT8  
\*pt)  
*XTS-AES-256 Decryption.*
- void [XTS\\_AES\\_256\\_dec\\_expanded\\_key](#) (UINT8 \*k2, UINT8 \*k1, UINT8 \*TW\_initial, UINT64 N, const UI-  
NT8 \*ct, UINT8 \*pt)  
*XTS-AES-256 Decryption with pre-expanded keys.*

### 7.1.1 Detailed Description

AES XTS encryption function prototypes. This defines the interface to optimized AES XTS functions

#### Pre-expanded keys

For key encryption, pre-expanded keys are stored in the order that they will be used. As an example, if Key[0] is the 128-bit initial key used for an AES-128 encryption, the rest of the keys are stored as follows:

- Key[0] : Initial encryption key
- Key[1] : Round 1 encryption key
- Key[2] : Round 2 encryption key
- ...
- Key[10] : Round 10 encryption key

For decryption, the order of keys does not change. However, we apply the necessary aesimc instructions before storing the expanded keys. For the same key used above, the pre-expanded keys will be stored as follows:

- Key[0] : Initial encryption key
- Key[1] : aesimc(Round 1 encryption key)
- Key[2] : aesimc(Round 2 encryption key)
- ...
- Key[9] : aesimc(Round 9 encryption key)
- Key[10] : Round 10 encryption key

**Note:** The expanded key decryption requires a decryption key only for the block decryption step. The tweak step in the expanded key decryption requires the same expanded encryption key that is used in the expanded key encryption.

#### Input and Output Buffers

The input and output buffers can be overlapping as long as the output buffer pointer is not less than the input buffer pointer. If the two pointers are the same, then encryption/decryption will occur in-place.

#### Data Length

- The functions support data length of any bytes greater than or equal to 16 bytes.
  - Data length is a 64-bit value, which makes the largest possible data length  $2^{64} - 1$  bytes.
  - For data lengths from 0 to 15 bytes, the functions return without any error codes, without reading or writing any data.
  - The functions only support byte lengths, not bits.
-

**Initial Tweak**

The functions accept a 128-bit initial tweak value. The user is responsible for padding the initial tweak value to this length.

**Data Alignment**

The input and output buffers, keys, pre-expanded keys and initial tweak value are not required to be aligned to 16 bytes, any alignment works.

**7.1.2 Function Documentation****7.1.2.1 void aes\_keyexp\_128 ( UINT8 \* *key*, UINT8 \* *exp\_key\_enc*, UINT8 \* *exp\_key\_dec* )**

AES-128 key expansion for encryption and decryption.

**Requires** AES-NI

**Parameters**

<i>key</i>	input key for AES-128, 16 bytes
<i>exp_key_enc</i>	expanded encryption keys, 16*11 bytes
<i>exp_key_dec</i>	expanded decryption keys, 16*11 bytes

**7.1.2.2 void aes\_keyexp\_256 ( UINT8 \* *key*, UINT8 \* *exp\_key\_enc*, UINT8 \* *exp\_key\_dec* )**

AES-256 key expansion for encryption and decryption.

**Requires** AES-NI

**Parameters**

<i>key</i>	input key for AES-256, 16*2 bytes
<i>exp_key_enc</i>	expanded encryption keys, 16*15 bytes
<i>exp_key_dec</i>	expanded decryption keys, 16*15 bytes

**7.1.2.3 void XTS\_AES\_128\_dec ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *ct*, UINT8 \* *pt* )**

XTS-AES-128 Decryption.

**Requires** AES-NI

---

**Parameters**

<i>k2</i>	key used for tweaking, 16 bytes
<i>k1</i>	key used for decryption of tweaked ciphertext, 16 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>ct</i>	ciphertext sector input data
<i>pt</i>	plaintext sector output data

**7.1.2.4** void XTS\_AES\_128\_dec\_expanded\_key ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *ct*, UINT8 \* *pt* )

XTS-AES-128 Decryption with pre-expanded keys.

**Requires** AES-NI

**Parameters**

<i>k2</i>	expanded key used for tweaking, 16*11 bytes - encryption key is used
<i>k1</i>	expanded decryption key used for decryption of tweaked ciphertext, 16*11 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>ct</i>	ciphertext sector input data
<i>pt</i>	plaintext sector output data

**7.1.2.5** void XTS\_AES\_128\_enc ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *pt*, UINT8 \* *ct* )

XTS-AES-128 Encryption.

**Requires** AES-NI

**Parameters**

<i>k2</i>	key used for tweaking, 16 bytes
<i>k1</i>	key used for encryption of tweaked plaintext, 16 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>pt</i>	plaintext sector input data
<i>ct</i>	ciphertext sector output data

**7.1.2.6** void XTS\_AES\_128\_enc\_expanded\_key ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *pt*, UINT8 \* *ct* )

XTS-AES-128 Encryption with pre-expanded keys.

**Requires** AES-NI

#### Parameters

<i>k2</i>	expanded key used for tweaking, 16*11 bytes
<i>k1</i>	expanded key used for encryption of tweaked plaintext, 16*11 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>pt</i>	plaintext sector input data
<i>ct</i>	ciphertext sector output data

**7.1.2.7** void XTS\_AES\_256\_dec ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *ct*, UINT8 \* *pt* )

XTS-AES-256 Decryption.

**Requires** AES-NI

#### Parameters

<i>k2</i>	key used for tweaking, 16*2 bytes
<i>k1</i>	key used for decryption of tweaked ciphertext, 16*2 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>ct</i>	ciphertext sector input data
<i>pt</i>	plaintext sector output data

**7.1.2.8** void XTS\_AES\_256\_dec\_expanded\_key ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *ct*, UINT8 \* *pt* )

XTS-AES-256 Decryption with pre-expanded keys.

**Requires** AES-NI

**Parameters**

<i>k2</i>	expanded key used for tweaking, 16*15 bytes - encryption key is used
<i>k1</i>	expanded decryption key used for decryption of tweaked ciphertext, 16*15 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>ct</i>	ciphertext sector input data
<i>pt</i>	plaintext sector output data

**7.1.2.9** void XTS\_AES\_256\_enc ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *pt*, UINT8 \* *ct* )

XTS-AES-256 Encryption.

**Requires** AES-NI

**Parameters**

<i>k2</i>	key used for tweaking, 16*2 bytes
<i>k1</i>	key used for encryption of tweaked plaintext, 16*2 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>pt</i>	plaintext sector input data
<i>ct</i>	ciphertext sector output data

**7.1.2.10** void XTS\_AES\_256\_enc\_expanded\_key ( UINT8 \* *k2*, UINT8 \* *k1*, UINT8 \* *TW\_initial*, UINT64 *N*, const UINT8 \* *pt*, UINT8 \* *ct* )

XTS-AES-256 Encryption with pre-expanded keys.

**Requires** AES-NI

**Parameters**

<i>k2</i>	expanded key used for tweaking, 16*15 bytes
<i>k1</i>	expanded key used for encryption of tweaked plaintext, 16*15 bytes
<i>TW_initial</i>	initial tweak value, 16 bytes
<i>N</i>	sector size, in bytes
<i>pt</i>	plaintext sector input data
<i>ct</i>	ciphertext sector output data



## 7.2 crc.h File Reference

CRC functions.

```
#include "types.h"
```

### Functions

- `UINT16 crc16_t10dif_01` (`UINT16 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the T10 standard.*
- `UINT16 crc16_t10dif_by4` (`UINT16 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the T10 standard. Optimized for SLM.*
- `UINT16 crc16_t10dif` (`UINT16 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the T10 standard, runs appropriate version.*
- `UINT32 crc32_ieee_01` (`UINT32 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the IEEE standard.*
- `UINT32 crc32_ieee_by4` (`UINT32 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the IEEE standard. Optimized for SLM.*
- `UINT32 crc32_ieee` (`UINT32 init_crc`, `const unsigned char *buf`, `UINT64 len`)  
*Generate CRC from the IEEE standard, runs appropriate version.*
- `unsigned int crc32_iscsi_simple` (`unsigned char *buffer`, `int len`, `unsigned int init_crc`)  
*ISCSI CRC simple implementation with CRC32 instruction.*
- `unsigned int crc32_iscsi_baseline` (`unsigned char *buffer`, `int len`, `unsigned int init_crc`)  
*ISCSI CRC baseline implementation with CRC32 instruction.*
- `unsigned int crc32_iscsi_00` (`unsigned char *buffer`, `int len`, `unsigned int init_crc`)  
*ISCSI CRC function optimized for Nehalem.*
- `unsigned int crc32_iscsi_01` (`unsigned char *buffer`, `int len`, `unsigned int init_crc`)  
*ISCSI CRC function optimized for Westmere.*
- `unsigned int crc32_iscsi` (`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 crc16_t10dif_base` (`UINT16 seed`, `UINT8 *buf`, `UINT64 len`)  
*Generate CRC from the T10 standard, runs baseline version.*
- `UINT32 crc32_ieee_base` (`UINT32 seed`, `UINT8 *buf`, `UINT64 len`)  
*Generate CRC from the IEEE standard, runs baseline version.*

### 7.2.1 Detailed Description

CRC functions.

---

## 7.2.2 Function Documentation

### 7.2.2.1 `UINT16 crc16_t10dif ( UINT16 init_crc, const unsigned char * buf, UINT64 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

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

#### Examples:

[crc\\_simple\\_test.c](#).

### 7.2.2.2 `UINT16 crc16_t10dif_01 ( UINT16 init_crc, const unsigned char * buf, UINT64 len )`

Generate CRC from the T10 standard.

**Requires** SSE3, CLMUL

#### Returns

16 bit CRC

#### Parameters

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

### 7.2.2.3 `UINT16 crc16_t10dif_base ( UINT16 seed, UINT8 * buf, UINT64 len )`

Generate CRC from the T10 standard, runs baseline version.

---

**Returns**

16 bit CRC

**Parameters**

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

**7.2.2.4** `UINT16 crc16_t10dif_by4 ( UINT16 init_crc, const unsigned char * buf, UINT64 len )`

Generate CRC from the T10 standard. Optimized for SLM.

**Requires** SSE4, PCLMULQDQ.

**Returns**

16 bit CRC

**Parameters**

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

**7.2.2.5** `UINT32 crc32_ieee ( UINT32 init_crc, const unsigned char * buf, UINT64 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.

**Returns**

32 bit CRC

**Parameters**

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

---

**Examples:**

[crc\\_simple\\_test.c](#).

**7.2.2.6** `UINT32 crc32_ieee_01 ( UINT32 init_crc, const unsigned char * buf, UINT64 len )`

Generate CRC from the IEEE standard.

**Requires** SSE3, CLMUL

**Returns**

32 bit CRC

**Parameters**

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

**7.2.2.7** `UINT32 crc32_ieee_base ( UINT32 seed, UINT8 * buf, UINT64 len )`

Generate CRC from the IEEE standard, runs baseline version.

**Returns**

32 bit CRC

**Parameters**

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

**7.2.2.8** `UINT32 crc32_ieee_by4 ( UINT32 init_crc, const unsigned char * buf, UINT64 len )`

Generate CRC from the IEEE standard.Optimized for SLM.

**Requires** SSE4, PCLMULQDQ.

---

**Returns**

32 bit CRC.

**Parameters**

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

**7.2.2.9 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**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

**7.2.2.10 unsigned int crc32\_iscsi\_00 ( unsigned char \* *buffer*, int *len*, unsigned int *init\_crc* )**

ISCSI CRC function optimized for Nehalem.

**Requires** SSE4.2

**Returns**

32 bit CRC

**Parameters**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

---

**7.2.2.11 unsigned int crc32\_iscsi\_01 ( unsigned char \* *buffer*, int *len*, unsigned int *init\_crc* )**

ISCSI CRC function optimized for Westmere.

**Requires** SSE4.2, CLMUL

**Returns**

32 bit CRC

**Parameters**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

**7.2.2.12 unsigned int crc32\_iscsi\_base ( unsigned char \* *buffer*, int *len*, unsigned int *crc\_init* )**

ISCSI CRC function, baseline version.

**Returns**

32 bit CRC

**Parameters**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>crc_init</i>	initial CRC value

**7.2.2.13 unsigned int crc32\_iscsi\_baseline ( unsigned char \* *buffer*, int *len*, unsigned int *init\_crc* )**

ISCSI CRC baseline implementation with CRC32 instruction.

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

**Requires** SSE4.2

**Returns**

32 bit CRC

---

**Parameters**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

**7.2.2.14 unsigned int crc32\_iscsi\_simple ( unsigned char \* *buffer*, int *len*, unsigned int *init\_crc* )**

ISCSI CRC simple implementation with CRC32 instruction.

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

**Requires** SSE4.2

**Returns**

32 bit CRC

**Parameters**

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

**7.3 erasure\_code.h File Reference**

Interface to functions supporting erasure code encode and decode.

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

**Functions**

- void [ec\\_init\\_tables](#) (int k, int rows, unsigned char \*a, unsigned char \*gftbls)  
*Initialize tables for fast Erasure Code encode and decode.*
  - void [ec\\_encode\\_data](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data, runs appropriate version.*
  - void [ec\\_encode\\_data\\_sse](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data.*
  - void [ec\\_encode\\_data\\_avx](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)
-

*Generate or decode erasure codes on blocks of data.*

- void [ec\\_encode\\_data\\_avx2](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)

*Generate or decode erasure codes on blocks of data.*

- void [ec\\_encode\\_data\\_base](#) (int len, int srcs, int dests, unsigned char \*v, unsigned char \*\*src, unsigned char \*\*dest)

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

- void [ec\\_encode\\_data\\_update](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)

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

- void [ec\\_encode\\_data\\_update\\_sse](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)

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

- void [ec\\_encode\\_data\\_update\\_avx](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)

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

- void [ec\\_encode\\_data\\_update\\_avx2](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)

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

- void [ec\\_encode\\_data\\_update\\_base](#) (int len, int k, int rows, int vec\_i, unsigned char \*v, unsigned char \*data, unsigned char \*\*dest)

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

- void [gf\\_vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_2vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_2vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_2vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_3vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with three outputs.*



- void [gf\\_3vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with three outputs.*
  - void [gf\\_3vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with three outputs.*
  - void [gf\\_4vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with four outputs.*
  - void [gf\\_4vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with four outputs.*
  - void [gf\\_4vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with four outputs.*
  - void [gf\\_5vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with five outputs.*
  - void [gf\\_5vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with five outputs.*
  - void [gf\\_5vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with five outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_vect\\_dot\\_prod\\_base](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector dot product, runs baseline version.*
  - void [gf\\_vect\\_dot\\_prod](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector dot product, runs appropriate version.*
  - void [gf\\_vect\\_mad](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, runs appropriate version.*
  - void [gf\\_vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
-

- void [gf\\_vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
  - void [gf\\_vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
  - void [gf\\_vect\\_mad\\_base](#) (int len, int vec, int vec\_i, unsigned char \*v, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, baseline version.*
  - void [gf\\_2vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. SSE version.*
  - void [gf\\_2vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. AVX version of [gf\\_2vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_2vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. AVX2 version of [gf\\_2vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_3vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. SSE version.*
  - void [gf\\_3vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX version of [gf\\_3vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_3vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX2 version of [gf\\_3vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_4vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. SSE version.*
  - void [gf\\_4vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX version of [gf\\_4vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_4vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX2 version of [gf\\_4vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_5vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. SSE version.*
  - void [gf\\_5vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. AVX version.*
-

- void `gf_5vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. AVX2 version.*
- void `gf_6vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. SSE version.*
- void `gf_6vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX version.*
- void `gf_6vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX2 version.*
- unsigned char `gf_mul` (unsigned char a, unsigned char b)  
*Single element GF(2<sup>8</sup>) multiply.*
- unsigned char `gf_inv` (unsigned char a)  
*Single element GF(2<sup>8</sup>) 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<sup>8</sup>)*

### 7.3.1 Detailed Description

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

### 7.3.2 Function Documentation

**7.3.2.1** void `ec_encode_data` ( int len, int k, int rows, unsigned char \* gftbbs, 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<sup>8</sup>) 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

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

#### Returns

none

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

Generate or decode erasure codes on blocks of data.

Arch specific version of [ec\\_encode\\_data\(\)](#) with same parameters.

**Requires** AVX

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

Generate or decode erasure codes on blocks of data.

Arch specific version of [ec\\_encode\\_data\(\)](#) with same parameters.

**Requires** AVX2

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

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

Baseline version of [ec\\_encode\\_data\(\)](#) with same parameters.

---

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

Generate or decode erasure codes on blocks of data.

Arch specific version of `ec_encode_data()` with same parameters.

**Requires** SSE4.1

**7.3.2.6** void `ec_encode_data_update` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *g\_tbls*, unsigned char \* *data*, unsigned char \*\* *coding* )

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

Given one source data block, update one or multiple blocks of encoded data as specified by a matrix of GF(2<sup>8</sup>) 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

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

#### Returns

none

**7.3.2.7** void `ec_encode_data_update_avx` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *g\_tbls*, unsigned char \* *data*, unsigned char \*\* *coding* )

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

Arch specific version of `ec_encode_data_update()` with same parameters.

**Requires** AVX

**7.3.2.8** void `ec_encode_data_update_avx2` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *g\_tbls*, unsigned char \* *data*, unsigned char \*\* *coding* )

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

Arch specific version of `ec_encode_data_update()` with same parameters.

**Requires** AVX2

**7.3.2.9** void `ec_encode_data_update_base` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *v*, unsigned char \* *data*, unsigned char \*\* *dest* )

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

Baseline version of `ec_encode_data_update()`.

**7.3.2.10** void `ec_encode_data_update_sse` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *g\_tbls*, unsigned char \* *data*, unsigned char \*\* *coding* )

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

Arch specific version of `ec_encode_data_update()` with same parameters.

**Requires** SSE4.1

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

Initialize tables for fast Erasure Code encode and decode.

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

#### Parameters

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

#### Returns

none

**7.3.2.12** `void gf_2vect_dot_prod_avx ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

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

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

**Requires** AVX

#### Parameters

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to $2*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

#### Returns

none

**7.3.2.13** `void gf_2vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

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

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

**Requires** AVX2

#### Parameters

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to $2*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

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

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

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

**Requires** SSE4.1

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbls</i>	Pointer to $2 \times 32 \times \text{vlen}$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

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

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

**Requires** AVX

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

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

**Requires** AVX2



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

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

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

**Requires** SSE4.1

#### Parameters

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vec</i>	The number of vector sources or rows in the generator matrix for coding.
<i>vec_i</i>	The vector index corresponding to the single input source.
<i>gftbls</i>	Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size $32 \times \text{vec}$ .
<i>src</i>	Pointer to source input array.
<i>dest</i>	Array of pointers to destination input/outputs.

#### Returns

none

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

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

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

**Requires** AVX

#### Parameters

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbls</i>	Pointer to $3 \times 32 \times \text{vlen}$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product with three outputs.

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

**Requires** AVX2

**Parameters**

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

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product with three outputs.

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

**Requires** SSE4.1

**Parameters**

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

**Returns**

none

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

GF( $2^8$ ) vector multiply with 3 accumulate. AVX version of [gf\\_3vect\\_mad\\_sse\(\)](#).

**Requires** AVX

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

GF( $2^8$ ) vector multiply with 3 accumulate. AVX2 version of [gf\\_3vect\\_mad\\_sse\(\)](#).

**Requires** AVX2

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

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

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

**Requires** SSE4.1

---

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vec</i>	The number of vector sources or rows in the generator matrix for coding.
<i>vec_i</i>	The vector index corresponding to the single input source.
<i>gftbls</i>	Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size $32 \times \text{vec}$ .
<i>src</i>	Pointer to source input array.
<i>dest</i>	Array of pointers to destination input/outputs.

**Returns**

none

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

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

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

**Requires** AVX

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbls</i>	Pointer to $4 \times 32 \times \text{vlen}$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

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

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

---

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

**Requires** AVX2

#### Parameters

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

#### Returns

none

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

GF(2<sup>8</sup>) vector dot product with four outputs.

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

**Requires** SSE4.1

#### Parameters

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

**Returns**

none

**7.3.2.27** void `gf_4vect_mad_avx` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX version of [gf\\_4vect\\_mad\\_sse\(\)](#).

**Requires** AVX

**7.3.2.28** void `gf_4vect_mad_avx2` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX2 version of [gf\\_4vect\\_mad\\_sse\(\)](#).

**Requires** AVX2

**7.3.2.29** void `gf_4vect_mad_sse` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 4 accumulate. SSE version.

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

**Requires** SSE4.1

**Parameters**

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

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product with five outputs.

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

**Requires** AVX

**Parameters**

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

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product with five outputs.

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

**Requires** AVX2

**Parameters**

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

**Returns**

none

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

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

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

**Requires** SSE4.1

**Parameters**

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

**Returns**

none

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

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

**Requires** AVX



**7.3.2.34** void `gf_5vect_mad_avx2` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 5 accumulate. AVX2 version.

**Requires** AVX2

**7.3.2.35** void `gf_5vect_mad_sse` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 5 accumulate. SSE version.

**Requires** SSE4.1

**7.3.2.36** void `gf_6vect_dot_prod_avx` ( int *len*, int *vlen*, unsigned char \* *gftbbs*, unsigned char \*\* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector dot product with six outputs.

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

**Requires** AVX

#### Parameters

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

**7.3.2.37** `void gf_6vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with six outputs.

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

**Requires** AVX2

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

**7.3.2.38** `void gf_6vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with six outputs.

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

**Requires** SSE4.1

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to $6 \times 32 \times vlen$ byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Array of pointers to destination data buffers.

**Returns**

none

**7.3.2.39** void gf\_6vect\_mad\_avx ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \*\* *dest* )

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

**Requires** AVX

**7.3.2.40** void gf\_6vect\_mad\_avx2 ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \*\* *dest* )

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

**Requires** AVX2

**7.3.2.41** void gf\_6vect\_mad\_sse ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \*\* *dest* )

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

**Requires** SSE4.1

**7.3.2.42** void gf\_gen\_cauchy1\_matrix ( unsigned char \* *a*, int *m*, int *k* )

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

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

**Parameters**

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

**Returns**

none

**7.3.2.43 void gf\_gen\_rs\_matrix ( unsigned char \* *a*, int *m*, int *k* )**

Generate a matrix of coefficients to be used for encoding.

Vandermonde matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as  $2^{i*(j-k+1)}$   $i:\{0,k-1\}$   $j:\{k,m-1\}$ . Commonly used method for choosing coefficients in erasure encoding but does not guarantee invertable for every sub matrix. For large k it is possible to find cases where the decode matrix chosen from sources and parity not in erasure are not invertable. Users may want to adjust for  $k > 5$ .

**Parameters**

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

**Returns**

none

**7.3.2.44 unsigned char gf\_inv ( unsigned char *a* )**

Single element GF( $2^8$ ) inverse.

**Parameters**

<i>a</i>	Input element
----------	---------------

**Returns**

Field element b such that  $a \times b = \{1\}$

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

Invert a matrix in GF(2<sup>8</sup>)

**Parameters**

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

**Returns**

0 successful, other fail on singular input matrix

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

Single element GF(2<sup>8</sup>) multiply.

**Parameters**

<i>a</i>	Multiplicand a
<i>b</i>	Multiplicand b

**Returns**

Product of a and b in GF(2<sup>8</sup>)

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

GF(2<sup>8</sup>) vector dot product, runs appropriate version.

Does a GF(2<sup>8</sup>) 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**

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

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product.

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

**Requires** AVX

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

**Returns**

none

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

GF(2<sup>8</sup>) vector dot product.

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

**Requires** AVX2

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

**Returns**

none

**7.3.2.50** `void gf_vect_dot_prod_base ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char * dest )`

GF(2<sup>8</sup>) vector dot product, runs baseline version.

Does a GF(2<sup>8</sup>) 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**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	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.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

**Returns**

none

**7.3.2.51** `void gf_vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char * dest )`

GF(2<sup>8</sup>) vector dot product.

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

**Requires** SSE4.1

**Parameters**

<i>len</i>	Length of each vector in bytes. Must be $\geq 16$ .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

**Returns**

none

**7.3.2.52** void `gf_vect_mad` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \* *dest* )

GF(2<sup>8</sup>) vector multiply accumulate, runs appropriate version.

Does a GF(2<sup>8</sup>) 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**

<i>len</i>	Length of each vector in bytes. Must be $\geq 32$ .
<i>vec</i>	The number of vector sources or rows in the generator matrix for coding.
<i>vec_i</i>	The vector index corresponding to the single input source.
<i>gftbbs</i>	Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size 32*vec.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

**Returns**

none

**7.3.2.53** void `gf_vect_mad_avx` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \* *dest* )

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of [gf\\_vect\\_mad\(\)](#) with same parameters.

**Requires** AVX

**7.3.2.54** void `gf_vect_mad_avx2` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbbs*, unsigned char \* *src*, unsigned char \* *dest* )

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of [gf\\_vect\\_mad\(\)](#) with same parameters.

**Requires** AVX2

---



**7.3.2.55** void gf\_vect\_mad\_base ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *v*, unsigned char \* *src*, unsigned char \* *dest* )

GF(2<sup>8</sup>) vector multiply accumulate, baseline version.

Baseline version of [gf\\_vect\\_mad\(\)](#) with same parameters.

**7.3.2.56** void gf\_vect\_mad\_sse ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \* *dest* )

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of [gf\\_vect\\_mad\(\)](#) with same parameters.

**Requires** SSE4.1

## 7.4 gf\_vect\_mul.h File Reference

Interface to functions for vector (block) multiplication in GF(2<sup>8</sup>).

### Functions

- int [gf\\_vect\\_mul\\_sse](#) (int *len*, unsigned char \**gftbl*, void \**src*, void \**dest*)  
*GF(2<sup>8</sup>) vector multiply by constant.*
- int [gf\\_vect\\_mul\\_avx](#) (int *len*, unsigned char \**gftbl*, void \**src*, void \**dest*)  
*GF(2<sup>8</sup>) vector multiply by constant.*
- int [gf\\_vect\\_mul](#) (int *len*, unsigned char \**gftbl*, void \**src*, void \**dest*)  
*GF(2<sup>8</sup>) 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<sup>8</sup>) vector multiply.*
- void [gf\\_vect\\_mul\\_base](#) (int *len*, unsigned char \**a*, unsigned char \**src*, unsigned char \**dest*)  
*GF(2<sup>8</sup>) vector multiply by constant, runs baseline version.*

### 7.4.1 Detailed Description

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

## 7.4.2 Function Documentation

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

GF(2<sup>8</sup>) vector multiply by constant, runs appropriate version.

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

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

#### Parameters

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

#### Returns

0 pass, other fail

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

GF(2<sup>8</sup>) vector multiply by constant.

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

**Requires** AVX

#### Parameters

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

#### Returns

0 pass, other fail

**7.4.2.3 void gf\_vect\_mul\_base ( int *len*, unsigned char \* *a*, unsigned char \* *src*, unsigned char \* *dest* )**

GF(2<sup>8</sup>) vector multiply by constant, runs baseline version.

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

**Parameters**

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

**7.4.2.4 void gf\_vect\_mul\_init ( unsigned char *c*, unsigned char \* *gftbl* )**

Initialize 32-byte constant array for GF(2<sup>8</sup>) vector multiply.

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

**Parameters**

<i>c</i>	Constant input.
<i>gftbl</i>	Table output.

**7.4.2.5 int gf\_vect\_mul\_sse ( int *len*, unsigned char \* *gftbl*, void \* *src*, void \* *dest* )**

GF(2<sup>8</sup>) vector multiply by constant.

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

**Requires** SSE4.1

**Parameters**

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

### Returns

0 pass, other fail

## 7.5 `igzip_lib.h` File Reference

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

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

### Data Structures

- struct `BitBuf2`  
*Holds Bit Buffer information.*
- struct `LZ_State1`  
*Holds the internal state information for input and output compression streams.*
- struct `LZ_Stream1`  
*Holds stream information.*

### Typedefs

- typedef struct `LZ_State1` `LZ_State1`  
*Holds the internal state information for input and output compression streams.*
- typedef struct `LZ_Stream1` `LZ_Stream1`  
*Holds stream information.*

### Enumerations

- enum `LZ_State1_state` { `LZS2_HDR`, `LZS2_BODY`, `LZS2_TRL`, `LZS2_END` }  
*Compression State please note `LZS2_TRL` only applies for GZIP compression.*

### Functions

- void `init_stream` (`LZ_Stream1` \*stream)  
*Initialize compression stream data structure.*
  - int `fast_lz` (`LZ_Stream1` \*stream)  
*Fast data (deflate) compression for storage applications.*
  - int `fast_lz_stateless` (`LZ_Stream1` \*stream)  
*Fast data (deflate) stateless compression for storage applications.*
-

### 7.5.1 Detailed Description

This file defines the `igzip` compression 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:

- Sync flush: whereby each call to `fast_lz` returns a new deflate block. Each deflate block is byte aligned with an empty stored block that is appended to the compressed output. With this flush mode an accurate bytes consumed figure is reported in the compression state.
- Finish Flush: whereby a call or multiple calls to `fast_lz` will return a single deflate block that is not byte aligned. Accurate bytes consumed is not supported with this flush mode

There are 4 major versions selectable at build time:

- `IGZIP0C`: the default version, it uses `PCLMULQDQ` for CRC calculations and 1 pointer in the hash table.
- `IGZIP1C`: it uses `PCLMULQDQ` for CRC calculations and a fixed array of 4 pointers in the hash table.
- `IGZIP0`: similar to `IGZIP0C`, with the `CLMUL` requirement no longer necessary.
- `IGZIP1`: similar to `IGZIP1C`, with the `CLMUL` requirement no longer necessary, and no limit on the hash update.

A number of configuration options are available, and can be used and combined to override `igzip`'s defaults. `igzip` default configuration is:

- 8K window size
- `IGZIP0C` major version

These options can be overridden to enable:

- 32K window size, a large window size, by adding `#define LARGE_WINDOW 1` in `igzip_lib.h` and `%define LARGE_WINDOW 1` in `options.inc`, or via the command line with

```
gmake D="LARGE_WINDOW=1"
```

on Linux and FreeBSD, or with

```
nmake -f Makefile.nmake D="-D LARGE_WINDOW=1"
```

on Windows.

- A different `igzip` major version, by passing a variable via command line with the version to select, such as

```
gmake D="MAJOR_VERSION=IGZIP0C"
```

on Linux and FreeBSD, and

---

```
nmake -f Makefile.nmake D="-D MAJOR_VERSION=IGZIP0C"
```

on Windows.

#### KNOWN ISSUES:

- Minimum size output buffer needs to be >218 Bytes, which is the size of the deflate header and trees.
- If building the code on Windows with the 32K window enabled, the `/LARGEADDRESSAWARE:NO` link option must be added.
- The 32K window isn't supported when used in a shared library.

### 7.5.2 Enumeration Type Documentation

#### 7.5.2.1 `enum LZ_State1_state`

Compression State please note `LZS2_TRL` only applies for GZIP compression.

##### Enumerator

**`LZS2_HDR`** Header state.

**`LZS2_BODY`** Body state.

**`LZS2_TRL`** Trailer state.

**`LZS2_END`** End state.

### 7.5.3 Function Documentation

#### 7.5.3.1 `int fast_lz ( LZ_Stream1 * stream )`

Fast data (deflate) compression for storage applications.

On entry to `fast_lz()`, `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 `fast_lz()`. These reflect the total number of bytes read or written so far.

The call to `fast_lz()` will take data from the input buffer (updating `next_in`, `avail_in` and in the case of sync flush `bytes_consumed` for accurate bytes consumed) and write a compressed stream to the output buffer (updating `next_out` and `avail_out`). Without sync flushing the function returns when either `avail_in` or `avail_out` goes to zero (i.e. when it runs out of input data or when the output buffer fills up, whichever comes first), producing one contiguous deflate block.

With sync flushing the function returns when it runs out of input data (`bytes_consumed` equals what was submitted in `avail_in`) or if it runs out of space (gets within 13 bytes from the end of the output buffer, `avail_out` <= 13bytes), whichever comes first. It produces one raw deflate block for each input buffer followed by an empty stored block

---

(sync flush per input buffer). When a buffer is submitted, it is copied into the internal state and `avail_in` is decremented to 0 (the internal state manages the offsets on each successive call to `fast_lz()`). The `bytes_consumed` variable will reflect how much of that input buffer has been compressed (for example: if there was not enough space in the output buffer). NOTE: `bytes_consumed` indicates exactly what was consumed from the input buffer even if `avail_in` returns as 0; `avail_in` needs to be updated if the input buffer was not fully consumed and the stream is re-initialized.

When the last input buffer is passed in, NOTE: the `end_of_stream` flag should be set (`FINISH_FLUSH` does not indicate this is the last buffer). This will cause the routine to complete the bit stream when it gets to the end of that input buffer, as long as the output buffer is big enough.

The equivalent of the zlib `FLUSH_SYNC` operation is currently supported. Flush types can be `SYNC_FLUSH` or `FINISH_FLUSH`. Default value is `FINISH_FLUSH`. if `SYNC_FLUSH` is selected each input buffer is compressed and byte aligned with a type 0 block appended to the end.

**Requires** SSE4.1, CLMUL

#### Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

#### Returns

`COMP_OK` (if everything is ok), `INVALID_FLUSH` (if an invalid FLUSH is selected), `INVALID_PARAM` (If `FLUSH_SYNC` is selected after `FLUSH_FINISH` without resetting the stream).

#### Examples:

[igzip\\_example.c](#).

#### 7.5.3.2 int fast\_lz\_stateless ( LZ\_Stream1 \* *stream* )

Fast data (deflate) stateless compression for storage applications.

Stateless (one shot) compression routine with a similar interface to `fast_lz()` 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.

**Requires** SSE4.1, CLMUL

#### Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

**Returns**

COMP\_OK (if everything is ok), STATELESS\_OVERFLOW (if output buffer will not fit output).

**7.5.3.3 void init\_stream ( LZ\_Stream1 \* *stream* )**

Initialize compression stream data structure.

**Requires** SSE4.1, CLMUL

**Parameters**

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

**Returns**

none

**Examples:**

[igzip\\_example.c](#).

## 7.6 intrinreg.h File Reference

Defines intrinsic types used by the new hashing API.

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

### 7.6.1 Detailed Description

Defines intrinsic types used by the new hashing API.

## 7.7 mb\_md5.h File Reference

Multi-buffer MD5 function prototypes and structures to submit jobs.

```
#include "types.h"
#include "multi_buffer.h"
```

---



## Data Structures

- struct [MD5\\_ARGS\\_X8](#)  
*Holds arguments for submitted MD5 job.*
- struct [MD5\\_ARGS\\_X8X2](#)  
*Holds arguments for submitted AVX2 MD5 job.*
- struct [JOB\\_MD5](#)  
*Holds info describing a single MD5 job for the multi-buffer manager.*
- struct [MD5\\_HMAC\\_LANE\\_DATA](#)  
*MD5 out-of-order scheduler fields.*
- struct [MD5\\_MB\\_MGR](#)  
*Holds state for multi-buffer MD5 jobs.*
- struct [MD5\\_MB\\_MGR\\_X8X2](#)  
*Holds state for multi-buffer AVX2 MD5 jobs.*

## Functions

- void [md5\\_init\\_mb\\_mgr](#) ([MD5\\_MB\\_MGR](#) \*state)  
*Initialize the MD5 multi-buffer manager structure.*
- void [md5\\_init\\_mb\\_mgr\\_x8x2](#) ([MD5\\_MB\\_MGR\\_X8X2](#) \*state)  
*Initialize the MD5 multi-buffer manager structure.*
- [JOB\\_MD5](#) \* [md5\\_submit\\_job](#) ([MD5\\_MB\\_MGR](#) \*state, [JOB\\_MD5](#) \*job)  
*Submit a new MD5 job to the multi-buffer manager.*
- [JOB\\_MD5](#) \* [md5\\_flush\\_job](#) ([MD5\\_MB\\_MGR](#) \*state)  
*Finish all submitted MD5 jobs and return when complete.*
- [JOB\\_MD5](#) \* [md5\\_submit\\_job\\_avx](#) ([MD5\\_MB\\_MGR](#) \*state, [JOB\\_MD5](#) \*job)  
*Submit a new MD5 job to the multi-buffer manager.*
- [JOB\\_MD5](#) \* [md5\\_flush\\_job\\_avx](#) ([MD5\\_MB\\_MGR](#) \*state)  
*Finish all submitted MD5 jobs and return when complete.*
- [JOB\\_MD5](#) \* [md5\\_submit\\_job\\_avx2](#) ([MD5\\_MB\\_MGR\\_X8X2](#) \*state, [JOB\\_MD5](#) \*job)  
*Submit a new MD5 job to the multi-buffer manager.*
- [JOB\\_MD5](#) \* [md5\\_flush\\_job\\_avx2](#) ([MD5\\_MB\\_MGR\\_X8X2](#) \*state)  
*Finish all submitted MD5 jobs and return when complete.*

### 7.7.1 Detailed Description

Multi-buffer MD5 function prototypes and structures to submit jobs. Interface for multi-buffer MD5 functions.

#### Multi-buffer MD5 Init/Update/Finalize

The multi-buffer md5 interface includes the ability to submit complete buffers for hashing (with init and finalize steps) or jobs in the category of init only (do initialization but no finalize) or update (no init or finalize steps). The job must

---

specify the flags HASH\_MB\_FIRST and/or HASH\_MB\_LAST, or HASH\_MB\_NO\_FLAGS to specify between the types of jobs. Job types without HASH\_MB\_LAST must be submitted with size as a multiple of the fundamental block size of 64 bytes. Note: The update function is not yet available for the AVX2 versions, but the job flags and total length must still be set.

## 7.7.2 Function Documentation

### 7.7.2.1 JOB\_MD5\* md5\_flush\_job ( MD5\_MB\_MGR \* *state* )

Finish all submitted MD5 jobs and return when complete.

**Requires** SSE4.1

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.7.2.2 JOB\_MD5\* md5\_flush\_job\_avx ( MD5\_MB\_MGR \* *state* )

Finish all submitted MD5 jobs and return when complete.

**Requires** AVX

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.7.2.3 JOB\_MD5\* md5\_flush\_job\_avx2 ( MD5\_MB\_MGR\_X8X2 \* *state* )

Finish all submitted MD5 jobs and return when complete.

**Requires** AVX2

---

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.7.2.4 void md5\_init\_mb\_mgr ( MD5\_MB\_MGR \* *state* )**

Initialize the MD5 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.7.2.5 void md5\_init\_mb\_mgr\_x8x2 ( MD5\_MB\_MGR\_X8X2 \* *state* )**

Initialize the MD5 multi-buffer manager structure.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.7.2.6 JOB\_MD5\* md5\_submit\_job ( MD5\_MB\_MGR \* *state*, JOB\_MD5 \* *job* )**

Submit a new MD5 job to the multi-buffer manager.

**Requires** SSE4.1

---

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.7.2.7 JOB\_MD5\* md5\_submit\_job\_avx ( MD5\_MB\_MGR \* *state*, JOB\_MD5 \* *job* )**

Submit a new MD5 job to the multi-buffer manager.

**Requires** AVX

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.7.2.8 JOB\_MD5\* md5\_submit\_job\_avx2 ( MD5\_MB\_MGR\_X8X2 \* *state*, JOB\_MD5 \* *job* )**

Submit a new MD5 job to the multi-buffer manager.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

---

## Returns

NULL if no jobs complete or pointer to jobs structure.

## 7.8 mb\_sha1.h File Reference

Multi-buffer SHA1 function prototypes and structures to submit jobs.

```
#include "types.h"
#include "multi_buffer.h"
```

## Data Structures

- struct [SHA1\\_ARGS\\_X4](#)  
*Holds arguments for submitted SHA1 job.*
- struct [SHA1\\_ARGS\\_X8](#)  
*Holds arguments for submitted SHA1 job.*
- struct [JOB\\_SHA1](#)  
*Holds info describing a single SHA1 job for the multi-buffer manager.*
- struct [SHA1\\_HMAC\\_LANE\\_DATA](#)  
*SHA1 out-of-order scheduler fields.*
- struct [SHA1\\_MB\\_MGR](#)  
*Holds state for multi-buffer SHA1 jobs.*
- struct [SHA1\\_MB\\_MGR\\_X8](#)  
*Holds state for multi-buffer SHA1 jobs.*

## Functions

- void [sha1\\_init\\_mb\\_mgr](#) ([SHA1\\_MB\\_MGR](#) \*state)  
*Initialize the SHA1 multi-buffer manager structure.*
  - void [sha1\\_init\\_mb\\_mgr\\_x8](#) ([SHA1\\_MB\\_MGR\\_X8](#) \*state)  
*Initialize the SHA1 multi-buffer manager structure.*
  - [JOB\\_SHA1](#) \* [sha1\\_submit\\_job](#) ([SHA1\\_MB\\_MGR](#) \*state, [JOB\\_SHA1](#) \*job)  
*Submit a new SHA1 job to the multi-buffer manager.*
  - [JOB\\_SHA1](#) \* [sha1\\_flush\\_job](#) ([SHA1\\_MB\\_MGR](#) \*state)  
*Finish all submitted SHA1 jobs and return when complete.*
  - [JOB\\_SHA1](#) \* [sha1\\_submit\\_job\\_avx](#) ([SHA1\\_MB\\_MGR](#) \*state, [JOB\\_SHA1](#) \*job)  
*Submit a new SHA1 job to the multi-buffer manager.*
  - [JOB\\_SHA1](#) \* [sha1\\_flush\\_job\\_avx](#) ([SHA1\\_MB\\_MGR](#) \*state)  
*Finish all submitted SHA1 jobs and return when complete.*
-

- **JOB\_SHA1 \* sha1\_submit\_job\_avx2** (SHA1\_MB\_MGR\_X8 \*state, JOB\_SHA1 \*job)  
*Submit a new SHA1 job to the multi-buffer manager.*
- **JOB\_SHA1 \* sha1\_flush\_job\_avx2** (SHA1\_MB\_MGR\_X8 \*state)  
*Finish all submitted SHA1 jobs and return when complete.*

### 7.8.1 Detailed Description

Multi-buffer SHA1 function prototypes and structures to submit jobs. Interface for multi-buffer SHA1 functions.

#### Multi-buffer SHA1 Init/Update/Finalize

The multi-buffer sha1 interface includes the ability to submit complete buffers for hashing (with init and finalize steps) or jobs in the category of init only (do initialization but no finalize) or update (no init or finalize steps). The job must specify the flags HASH\_MB\_FIRST and/or HASH\_MB\_LAST, or HASH\_MB\_NO\_FLAGS to specify between the types of jobs. Job types without HASH\_MB\_LAST must be submitted with size as a multiple of the fundamental block size of 64 bytes. Note: The update function is not yet available for the AVX2 versions, but the job flags and total length must still be set.

### 7.8.2 Function Documentation

#### 7.8.2.1 JOB\_SHA1\* sha1\_flush\_job ( SHA1\_MB\_MGR \* state )

Finish all submitted SHA1 jobs and return when complete.

**Requires** SSE4.1

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

#### Examples:

[multi\\_buffer\\_sha1\\_example.c](#).

#### 7.8.2.2 JOB\_SHA1\* sha1\_flush\_job\_avx ( SHA1\_MB\_MGR \* state )

Finish all submitted SHA1 jobs and return when complete.

**Requires** AVX

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.8.2.3 JOB\_SHA1\* sha1\_flush\_job\_avx2 ( SHA1\_MB\_MGR\_X8 \* *state* )**

Finish all submitted SHA1 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.8.2.4 void sha1\_init\_mb\_mgr ( SHA1\_MB\_MGR \* *state* )**

Initialize the SHA1 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**Examples:**

[multi\\_buffer\\_sha1\\_example.c](#).

---

**7.8.2.5 void sha1\_init\_mb\_mgr\_x8 ( SHA1\_MB\_MGR\_X8 \* *state* )**

Initialize the SHA1 multi-buffer manager structure.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.8.2.6 JOB\_SHA1\* sha1\_submit\_job ( SHA1\_MB\_MGR \* *state*, JOB\_SHA1 \* *job* )**

Submit a new SHA1 job to the multi-buffer manager.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**Examples:**

[multi\\_buffer\\_sha1\\_example.c](#).

**7.8.2.7 JOB\_SHA1\* sha1\_submit\_job\_avx ( SHA1\_MB\_MGR \* *state*, JOB\_SHA1 \* *job* )**

Submit a new SHA1 job to the multi-buffer manager.

**Requires** AVX

**Parameters**

---



<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

### 7.8.2.8 JOB\_SHA1\* sha1\_submit\_job\_avx2 ( SHA1\_MB\_MGR\_X8 \* *state*, JOB\_SHA1 \* *job* )

Submit a new SHA1 job to the multi-buffer manager.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

## 7.9 mb\_sha256.h File Reference

Multi-buffer SHA256 function prototypes and structures to submit jobs.

```
#include "types.h"
#include "multi_buffer.h"
```

**Data Structures**

- struct [SHA256\\_ARGS\\_X4](#)  
*Holds arguments for submitted SHA256 job.*
  - struct [SHA256\\_ARGS\\_X8](#)  
*Holds arguments for submitted SHA256 job.*
  - struct [JOB\\_SHA256](#)  
*Holds info describing a single SHA256 job for the multi-buffer manager.*
  - struct [SHA256\\_HMAC\\_LANE\\_DATA](#)
-

*SHA256 out-of-order scheduler fields.*

- struct [SHA256\\_MB\\_MGR](#)

*Holds state for multi-buffer SHA256 jobs.*

- struct [SHA256\\_MB\\_MGR\\_X8](#)

*Holds state for multi-buffer SHA256 jobs.*

## Functions

- void [sha256\\_init\\_mb\\_mgr](#) ([SHA256\\_MB\\_MGR](#) \*state)  
*Initialize the SHA256 multi-buffer manager structure.*
- void [sha256\\_init\\_mb\\_mgr\\_x8](#) ([SHA256\\_MB\\_MGR\\_X8](#) \*state)  
*Initialize the SHA256 multi-buffer manager structure.*
- [JOB\\_SHA256](#) \* [sha256\\_submit\\_job](#) ([SHA256\\_MB\\_MGR](#) \*state, [JOB\\_SHA256](#) \*job)  
*Submit a new SHA256 job to the multi-buffer manager.*
- [JOB\\_SHA256](#) \* [sha256\\_flush\\_job](#) ([SHA256\\_MB\\_MGR](#) \*state)  
*Finish all submitted SHA256 jobs and return when complete.*
- [JOB\\_SHA256](#) \* [sha256\\_submit\\_job\\_avx](#) ([SHA256\\_MB\\_MGR](#) \*state, [JOB\\_SHA256](#) \*job)  
*Submit a new SHA256 job to the multi-buffer manager.*
- [JOB\\_SHA256](#) \* [sha256\\_flush\\_job\\_avx](#) ([SHA256\\_MB\\_MGR](#) \*state)  
*Finish all submitted SHA256 jobs and return when complete.*
- [JOB\\_SHA256](#) \* [sha256\\_submit\\_job\\_avx2](#) ([SHA256\\_MB\\_MGR\\_X8](#) \*state, [JOB\\_SHA256](#) \*job)  
*Submit a new SHA256 job to the multi-buffer manager.*
- [JOB\\_SHA256](#) \* [sha256\\_flush\\_job\\_avx2](#) ([SHA256\\_MB\\_MGR\\_X8](#) \*state)  
*Finish all submitted SHA256 jobs and return when complete.*

### 7.9.1 Detailed Description

Multi-buffer SHA256 function prototypes and structures to submit jobs. Interface for multi-buffer SHA256 functions.

#### Multi-buffer SHA256 Init/Update/Finalize

The multi-buffer sha256 interface includes the ability to submit complete buffers for hashing (with init and finalize steps) or jobs in the category of init only (do initialization but no finalize) or update (no init or finalize steps). The job must specify the flags `HASH_MB_FIRST` and/or `HASH_MB_LAST`, or `HASH_MB_NO_FLAGS` to specify between the types of jobs. Job types without `HASH_MB_LAST` must be submitted with size as a multiple of the fundamental block size of 64 bytes. Note: The update function is not yet available for the AVX2 versions, but the job flags and total length must still be set.

## 7.9.2 Function Documentation

### 7.9.2.1 JOB\_SHA256\* sha256\_flush\_job ( SHA256\_MB\_MGR \* *state* )

Finish all submitted SHA256 jobs and return when complete.

**Requires** SSE4.1

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.9.2.2 JOB\_SHA256\* sha256\_flush\_job\_avx ( SHA256\_MB\_MGR \* *state* )

Finish all submitted SHA256 jobs and return when complete.

**Requires** AVX

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.9.2.3 JOB\_SHA256\* sha256\_flush\_job\_avx2 ( SHA256\_MB\_MGR\_X8 \* *state* )

Finish all submitted SHA256 jobs and return when complete.

**Requires** AVX2

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

---

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.9.2.4 void sha256\_init\_mb\_mgr ( SHA256\_MB\_MGR \* *state* )**

Initialize the SHA256 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.9.2.5 void sha256\_init\_mb\_mgr\_x8 ( SHA256\_MB\_MGR\_X8 \* *state* )**

Initialize the SHA256 multi-buffer manager structure.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.9.2.6 JOB\_SHA256\* sha256\_submit\_job ( SHA256\_MB\_MGR \* *state*, JOB\_SHA256 \* *job* )**

Submit a new SHA256 job to the multi-buffer manager.

**Requires** SSE4.1

---

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.9.2.7 JOB\_SHA256\* sha256\_submit\_job\_avx ( SHA256\_MB\_MGR \* *state*, JOB\_SHA256 \* *job* )**

Submit a new SHA256 job to the multi-buffer manager.

**Requires** AVX

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.9.2.8 JOB\_SHA256\* sha256\_submit\_job\_avx2 ( SHA256\_MB\_MGR\_X8 \* *state*, JOB\_SHA256 \* *job* )**

Submit a new SHA256 job to the multi-buffer manager.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

---

### Returns

NULL if no jobs complete or pointer to jobs structure.

## 7.10 mb\_sha512.h File Reference

Multi-buffer SHA512 function prototypes and structures to submit jobs.

```
#include "types.h"
#include "multi_buffer.h"
```

### Data Structures

- struct [SHA512\\_ARGS\\_X2](#)  
*Holds arguments for submitted SHA512 job.*
- struct [SHA512\\_ARGS\\_X4](#)  
*Holds arguments for submitted AVX2 SHA512 job.*
- struct [JOB\\_SHA512](#)  
*Holds info describing a single SHA512 job for the multi-buffer manager.*
- struct [SHA512\\_HMAC\\_LANE\\_DATA](#)  
*SHA512 out-of-order scheduler fields.*
- struct [SHA512\\_MB\\_MGR](#)  
*Holds state for multi-buffer SHA512 jobs.*
- struct [SHA512\\_MB\\_MGR\\_X4](#)  
*Holds state for multi-buffer SHA512 jobs.*

### Functions

- void [sha512\\_init\\_mb\\_mgr](#) (SHA512\_MB\_MGR \*state)  
*Initialize the SHA512 multi-buffer manager structure.*
  - void [sha512\\_init\\_mb\\_mgr\\_x4](#) (SHA512\_MB\_MGR\_X4 \*state)  
*Initialize the SHA512 multi-buffer manager structure.*
  - [JOB\\_SHA512 \\* sha512\\_submit\\_job](#) (SHA512\_MB\_MGR \*state, [JOB\\_SHA512 \\*job](#))  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [JOB\\_SHA512 \\* sha512\\_flush\\_job](#) (SHA512\_MB\_MGR \*state)  
*Finish all submitted SHA512 jobs and return when complete.*
  - [JOB\\_SHA512 \\* sha512\\_submit\\_job\\_avx](#) (SHA512\_MB\_MGR \*state, [JOB\\_SHA512 \\*job](#))  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [JOB\\_SHA512 \\* sha512\\_flush\\_job\\_avx](#) (SHA512\_MB\_MGR \*state)  
*Finish all submitted SHA512 jobs and return when complete.*
-

- `JOB_SHA512 * sha512_submit_job_avx2 (SHA512_MB_MGR_X4 *state, JOB_SHA512 *job)`  
*Submit a new SHA512 job to the multi-buffer manager.*
- `JOB_SHA512 * sha512_flush_job_avx2 (SHA512_MB_MGR_X4 *state)`  
*Finish all submitted SHA512 jobs and return when complete.*

### 7.10.1 Detailed Description

Multi-buffer SHA512 function prototypes and structures to submit jobs. Interface for multi-buffer SHA512 functions.

#### Multi-buffer SHA512 Init/Update/Finalize

The multi-buffer sha512 interface includes the ability to submit complete buffers for hashing (with init and finalize steps) or jobs in the category of init only (do initialization but no finalize) or update (no init or finalize steps). The job must specify the flags `HASH_MB_FIRST` and/or `HASH_MB_LAST`, or `HASH_MB_NO_FLAGS` to specify between the types of jobs. Job types without `HASH_MB_LAST` must be submitted with size as a multiple of the fundamental block size of 128 bytes. Note: The update function is not yet available for the AVX2 versions, but the job flags and total length must still be set.

### 7.10.2 Function Documentation

#### 7.10.2.1 `JOB_SHA512* sha512_flush_job ( SHA512_MB_MGR * state )`

Finish all submitted SHA512 jobs and return when complete.

**Requires** SSE4.1

#### Parameters

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

#### 7.10.2.2 `JOB_SHA512* sha512_flush_job_avx ( SHA512_MB_MGR * state )`

Finish all submitted SHA512 jobs and return when complete.

**Requires** AVX

#### Parameters

---

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.10.2.3 JOB\_SHA512\* sha512\_flush\_job\_avx2 ( SHA512\_MB\_MGR\_X4 \* *state* )**

Finish all submitted SHA512 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.10.2.4 void sha512\_init\_mb\_mgr ( SHA512\_MB\_MGR \* *state* )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.10.2.5 void sha512\_init\_mb\_mgr\_x4 ( SHA512\_MB\_MGR\_X4 \* *state* )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** AVX2

---



**Parameters**

<i>state</i>	Structure holding jobs state info
--------------	-----------------------------------

**Returns**

void

**7.10.2.6 JOB\_SHA512\* sha512\_submit\_job ( SHA512\_MB\_MGR \* *state*, JOB\_SHA512 \* *job* )**

Submit a new SHA512 job to the multi-buffer manager.

**Requires** SSE4.1

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.10.2.7 JOB\_SHA512\* sha512\_submit\_job\_avx ( SHA512\_MB\_MGR \* *state*, JOB\_SHA512 \* *job* )**

Submit a new SHA512 job to the multi-buffer manager.

**Requires** AVX

**Parameters**

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.10.2.8 JOB\_SHA512\* sha512\_submit\_job\_avx2 ( SHA512\_MB\_MGR\_X4 \* *state*, JOB\_SHA512 \* *job* )**

Submit a new SHA512 job to the multi-buffer manager.

---

**Requires** AVX2

#### Parameters

<i>state</i>	Structure holding jobs state info
<i>job</i>	Structure holding new job info

#### Returns

NULL if no jobs complete or pointer to jobs structure.

## 7.11 md5\_mb.h File Reference

Multi-buffer CTX API MD5 function prototypes and structures.

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

#### Data Structures

- struct [MD5\\_JOB](#)  
*Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.*
- struct [MD5\\_MB\\_ARGS\\_X16](#)  
*Scheduler layer - Holds arguments for submitted MD5 job.*
- struct [MD5\\_LANE\\_DATA](#)  
*Scheduler layer - Lane data.*
- struct [MD5\\_MB\\_JOB\\_MGR](#)  
*Scheduler layer - Holds state for multi-buffer MD5 jobs.*
- struct [MD5\\_HASH\\_CTX\\_MGR](#)  
*Context layer - Holds state for multi-buffer MD5 jobs.*
- struct [MD5\\_HASH\\_CTX](#)  
*Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.*

#### Functions

- void [md5\\_ctx\\_mgr\\_init\\_sse](#) ([MD5\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the context level MD5 multi-buffer manager structure.*
  - [MD5\\_HASH\\_CTX](#) \* [md5\\_ctx\\_mgr\\_submit\\_sse](#) ([MD5\\_HASH\\_CTX\\_MGR](#) \*mgr, [MD5\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)
-

*Submit a new MD5 job to the context level multi-buffer manager.*

- `MD5_HASH_CTX * md5_ctx_mgr_flush_sse (MD5_HASH_CTX_MGR *mgr)`

*Finish all submitted MD5 jobs and return when complete.*

- `void md5_ctx_mgr_init_avx (MD5_HASH_CTX_MGR *mgr)`

*Initialize the MD5 multi-buffer manager structure.*

- `MD5_HASH_CTX * md5_ctx_mgr_submit_avx (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)`

*Submit a new MD5 job to the multi-buffer manager.*

- `MD5_HASH_CTX * md5_ctx_mgr_flush_avx (MD5_HASH_CTX_MGR *mgr)`

*Finish all submitted MD5 jobs and return when complete.*

- `void md5_ctx_mgr_init_avx2 (MD5_HASH_CTX_MGR *mgr)`

*Initialize the MD5 multi-buffer manager structure.*

- `MD5_HASH_CTX * md5_ctx_mgr_submit_avx2 (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)`

*Submit a new MD5 job to the multi-buffer manager.*

- `MD5_HASH_CTX * md5_ctx_mgr_flush_avx2 (MD5_HASH_CTX_MGR *mgr)`

*Finish all submitted MD5 jobs and return when complete.*

- `void md5_ctx_mgr_init (MD5_HASH_CTX_MGR *mgr)`

*Initialize the MD5 multi-buffer manager structure.*

- `MD5_HASH_CTX * md5_ctx_mgr_submit (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)`

*Submit a new MD5 job to the multi-buffer manager.*

- `MD5_HASH_CTX * md5_ctx_mgr_flush (MD5_HASH_CTX_MGR *mgr)`

*Finish all submitted MD5 jobs and return when complete.*

### 7.11.1 Detailed Description

Multi-buffer CTX API MD5 function prototypes and structures. Interface for multi-buffer MD5 functions

#### Multi-buffer MD5 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the `MD5_HASH_CTX_MGR` and `MD5_HASH_CTX` objects. Numerous `MD5_HASH_CTX` objects may be instantiated by the application for use with a single `MD5_HASH_CTX_MGR`.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the `HASH_ENTIRE` flag, or as partial jobs which can be started using the `HASH_FIRST` flag, and later resumed or finished using the `HASH_UPDATE` and `HASH_LAST` flags respectively.

**Note:** The submit function does not require data buffers to be block sized.

The MD5 CTX interface functions are available for 3 architectures: SSE, AVX and AVX2. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

**Usage:** The application creates a [MD5\\_HASH\\_CTX\\_MGR](#) object and initializes it with a call to `md5_ctx_mgr_init*()` function, where henceforth "\*" stands for the relevant suffix for each architecture; `_sse`, `_avx`, `_avx2` (or no suffix for the multibinary version). The [MD5\\_HASH\\_CTX\\_MGR](#) object will be used to schedule processor resources, with up to 8 [MD5\\_HASH\\_CTX](#) objects (or 16 in the AVX2 case) being processed at a time.

Each [MD5\\_HASH\\_CTX](#) must be initialized before first use by the `hash_ctx_init` macro defined in [multi\\_buffer.h](#). After initialization, the application may begin computing a hash by giving the [MD5\\_HASH\\_CTX](#) to a [MD5\\_HASH\\_CTX\\_MGR](#) using the submit functions `md5_ctx_mgr_submit*()` with the `HASH_FIRST` flag set. When the [MD5\\_HASH\\_CTX](#) is returned to the application (via this or a later call to `md5_ctx_mgr_submit*()` or `md5_ctx_mgr_flush*()`), the application can then re-submit it with another call to `md5_ctx_mgr_submit*()`, but without the `HASH_FIRST` flag set.

Ideally, on the last buffer for that hash, `md5_ctx_mgr_submit_sse` is called with `HASH_LAST`, although it is also possible to submit the hash with `HASH_LAST` and a zero length if necessary. When a [MD5\\_HASH\\_CTX](#) is returned after having been submitted with `HASH_LAST`, it will contain a valid hash. The [MD5\\_HASH\\_CTX](#) can be reused immediately by submitting with `HASH_FIRST`.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: `HASH_FIRST | HASH_LAST` (or, equivalently, `HASH_ENTIRE`)
- two buffers: `HASH_FIRST`, `HASH_LAST`
- three buffers: `HASH_FIRST`, `HASH_UPDATE`, `HASH_LAST` etc.

The order in which `MD5_CTX` objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed `entire/first/update/last` values
- Submitting a context that is currently being managed by a [MD5\\_HASH\\_CTX\\_MGR](#).
- Submitting a context after `HASH_LAST` is used but before `HASH_FIRST` is set.

These error conditions are reported by returning the [MD5\\_HASH\\_CTX](#) immediately after a submit with its error member set to a non-zero error code (defined in [multi\\_buffer.h](#)). No changes are made to the [MD5\\_HASH\\_CTX\\_MGR](#) in the case of an error; no processing is done for other hashes.

## 7.11.2 Function Documentation

### 7.11.2.1 MD5\_HASH\_CTX\* md5\_ctx\_mgr\_flush ( MD5\_HASH\_CTX\_MGR \* mgr )

Finish all submitted MD5 jobs and return when complete.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.11.2.2 MD5\_HASH\_CTX\* md5\_ctx\_mgr\_flush\_avx ( MD5\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted MD5 jobs and return when complete.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.11.2.3 MD5\_HASH\_CTX\* md5\_ctx\_mgr\_flush\_avx2 ( MD5\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted MD5 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.11.2.4 MD5\_HASH\_CTX\* md5\_ctx\_mgr\_flush\_sse ( MD5\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted MD5 jobs and return when complete.

**Requires** SSE4.1

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.11.2.5 void md5\_ctx\_mgr\_init ( MD5\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the MD5 multi-buffer manager structure.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.11.2.6 void md5\_ctx\_mgr\_init\_avx ( MD5\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the MD5 multi-buffer manager structure.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.11.2.7 void md5\_ctx\_mgr\_init\_avx2 ( MD5\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the MD5 multi-buffer manager structure.

**Requires** AVX2

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.11.2.8 void md5\_ctx\_mgr\_init\_sse ( MD5\_HASH\_CTX\_MGR \* mgr )**

Initialize the context level MD5 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.11.2.9 MD5\_HASH\_CTX\* md5\_ctx\_mgr\_submit ( MD5\_HASH\_CTX\_MGR \* mgr, MD5\_HASH\_CTX \* ctx, const void \* buffer, uint32\_t len, HASH\_CTX\_FLAG flags )**

Submit a new MD5 job to the multi-buffer manager.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.11.2.10** MD5\_HASH\_CTX\* md5\_ctx\_mgr\_submit\_avx ( MD5\_HASH\_CTX\_MGR \* *mgr*, MD5\_HASH\_CTX \* *ctx*, const void \* *buffer*, uint32\_t *len*, HASH\_CTX\_FLAG *flags* )

Submit a new MD5 job to the multi-buffer manager.

**Requires** AVX

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.11.2.11** MD5\_HASH\_CTX\* md5\_ctx\_mgr\_submit\_avx2 ( MD5\_HASH\_CTX\_MGR \* *mgr*, MD5\_HASH\_CTX \* *ctx*, const void \* *buffer*, uint32\_t *len*, HASH\_CTX\_FLAG *flags* )

Submit a new MD5 job to the multi-buffer manager.

**Requires** AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.11.2.12** MD5\_HASH\_CTX\* md5\_ctx\_mgr\_submit\_sse ( MD5\_HASH\_CTX\_MGR \* *mgr*, MD5\_HASH\_CTX \* *ctx*, const void \* *buffer*, uint32\_t *len*, HASH\_CTX\_FLAG *flags* )

Submit a new MD5 job to the context level multi-buffer manager.

---



**Requires** SSE4.1

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

## 7.12 mem\_routines.h File Reference

Interface to storage mem operations.

### Functions

- int [mem\\_zero\\_detect\\_avx](#) (void \*mem, int len)  
*Detect if a memory region is all zero.*
- int [mem\\_cmp\\_sse](#) (void \*src, void \*des, int n)  
*Compare two memory blocks.*
- int [mem\\_cmp\\_avx](#) (void \*src, void \*des, int n)  
*Compare two memory blocks.*
- int [mem\\_cmp\\_avx2](#) (void \*src, void \*des, int n)  
*Compare two memory blocks.*
- void \* [mem\\_cpy\\_sse](#) (void \*des, void \*src, int n)  
*Copy memory blocks from src to des. Source and destination addresses cannot overlap.*
- void \* [mem\\_cpy\\_avx](#) (void \*des, void \*src, int n)  
*Copy memory blocks from src to des. Source and destination addresses cannot overlap.*

### 7.12.1 Detailed Description

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

---

## 7.12.2 Function Documentation

### 7.12.2.1 int mem\_cmp\_avx ( void \* *src*, void \* *des*, int *n* )

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 256 bytes

**Requires** AVX

#### Parameters

<i>src</i>	the first memory region
<i>des</i>	the second memory region
<i>n</i>	the length of each memory region in bytes

#### Returns

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

### 7.12.2.2 int mem\_cmp\_avx2 ( void \* *src*, void \* *des*, int *n* )

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 256 bytes

**Requires** AVX2

#### Parameters

<i>src</i>	the first memory region
<i>des</i>	the second memory region
<i>n</i>	the length of each memory region in bytes

#### Returns

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

### 7.12.2.3 int mem\_cmp\_sse ( void \* *src*, void \* *des*, int *n* )

Compare two memory blocks.

Memory compare function with optimizations for large blocks > 128 bytes

---

**Requires** SSE4.1

#### Parameters

<i>src</i>	the first memory region
<i>des</i>	the second memory region
<i>n</i>	the length of each memory region in bytes

#### Returns

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

#### 7.12.2.4 void\* mem\_cpy\_avx ( void \* *des*, void \* *src*, int *n* )

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

Memory copy function with optimizations for large blocks > 256 bytes

**Requires** AVX

#### Parameters

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

#### Returns

the start address of the destination memory region

#### 7.12.2.5 void\* mem\_cpy\_sse ( void \* *des*, void \* *src*, int *n* )

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

Memory copy function with optimizations for large blocks > 128 bytes

**Requires** SSE2

#### Parameters

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

**Returns**

the start address of the destination memory region

**7.12.2.6 int mem\_zero\_detect\_avx ( void \* *mem*, int *len* )**

Detect if a memory region is all zero.

Zero detect function with optimizations for large blocks > 128 bytes

**Requires** AVX

**Parameters**

<i>mem</i>	Pointer to memory region to test
<i>len</i>	Length of region in bytes

**Returns**

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

**7.13 memcpy\_inline.h File Reference**

Defines intrinsic memcpy functions used by the new hashing API.

```
#include "intrinreg.h"
#include <assert.h>
```

**7.13.1 Detailed Description**

Defines intrinsic memcpy functions used by the new hashing API.

**7.14 multi\_buffer.h File Reference**

Multi-buffer common fields.

**Enumerations**

- enum **JOB\_STS** {  
     **STS\_UNKNOWN** = 0, **STS\_BEING\_PROCESSED** = 1, **STS\_COMPLETED** = 2, **STS\_INTERNAL\_ERROR**,  
     **STS\_ERROR** }

*Job return codes.*

- enum `HASH_CTX_FLAG` { `HASH_UPDATE` = 0x00, `HASH_FIRST` = 0x01, `HASH_LAST` = 0x02, `HASH_ENTIRE` = 0x03 }

*CTX job type.*

- enum `HASH_CTX_STS` { `HASH_CTX_STS_IDLE` = 0x00, `HASH_CTX_STS_PROCESSING` = 0x01, `HASH_CTX_STS_LAST` = 0x02, `HASH_CTX_STS_COMPLETE` = 0x04 }

*CTX status flags.*

- enum `HASH_CTX_ERROR` { `HASH_CTX_ERROR_NONE` = 0, `HASH_CTX_ERROR_INVALID_FLAGS` = -1, `HASH_CTX_ERROR_ALREADY_PROCESSING` = -2, `HASH_CTX_ERROR_ALREADY_COMPLETED` = -3 }

*CTX error flags.*

### 7.14.1 Detailed Description

Multi-buffer common fields.

### 7.14.2 Enumeration Type Documentation

#### 7.14.2.1 enum HASH\_CTX\_ERROR

CTX error flags.

Enumerator

***HASH\_CTX\_ERROR\_NONE*** `HASH_CTX_ERROR_NONE`.

***HASH\_CTX\_ERROR\_INVALID\_FLAGS*** `HASH_CTX_ERROR_INVALID_FLAGS`.

***HASH\_CTX\_ERROR\_ALREADY\_PROCESSING*** `HASH_CTX_ERROR_ALREADY_PROCESSING`.

***HASH\_CTX\_ERROR\_ALREADY\_COMPLETED*** `HASH_CTX_ERROR_ALREADY_COMPLETED`.

#### 7.14.2.2 enum HASH\_CTX\_FLAG

CTX job type.

Enumerator

***HASH\_UPDATE*** `HASH_UPDATE`.

***HASH\_FIRST*** `HASH_FIRST`.

***HASH\_LAST*** `HASH_LAST`.

***HASH\_ENTIRE*** `HASH_ENTIRE`.

---

## 7.14.2.3 enum HASH\_CTX\_STS

CTX status flags.

## Enumerator

***HASH\_CTX\_STS\_IDLE*** HASH\_CTX\_STS\_IDLE.  
***HASH\_CTX\_STS\_PROCESSING*** HASH\_CTX\_STS\_PROCESSING.  
***HASH\_CTX\_STS\_LAST*** HASH\_CTX\_STS\_LAST.  
***HASH\_CTX\_STS\_COMPLETE*** HASH\_CTX\_STS\_COMPLETE.

## 7.14.2.4 enum JOB\_STS

Job return codes.

## Enumerator

***STS\_UNKNOWN*** STS\_UNKNOWN.  
***STS\_BEING\_PROCESSED*** STS\_BEING\_PROCESSED.  
***STS\_COMPLETED*** STS\_COMPLETED.  
***STS\_INTERNAL\_ERROR*** STS\_INTERNAL\_ERROR.  
***STS\_ERROR*** STS\_ERROR.

## 7.15 raid.h File Reference

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

## Functions

- int [xor\\_gen\\_sse](#) (int vects, int len, void \*\*array)  
*Generate XOR parity vector from N sources.*
  - int [xor\\_gen\\_avx](#) (int vects, int len, void \*\*array)  
*Generate XOR parity vector from N sources.*
  - int [xor\\_gen](#) (int vects, int len, void \*\*array)  
*Generate XOR parity vector from N sources, runs appropriate version.*
  - int [xor\\_check\\_sse](#) (int vects, int len, void \*\*array)  
*Checks that array has XOR parity sum of 0 across all vectors.*
  - 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\\_sse](#) (int vects, int len, void \*\*array)  
*Generate P+Q parity vectors from N sources.*
- int [pq\\_gen\\_avx](#) (int vects, int len, void \*\*array)  
*Generate P+Q parity vectors from N sources.*
- int [pq\\_gen\\_avx2](#) (int vects, int len, void \*\*array)  
*Generate P+Q parity vectors from N sources.*
- int [pq\\_gen](#) (int vects, int len, void \*\*array)  
*Generate P+Q parity vectors from N sources, runs appropriate version.*
- int [pq\\_check\\_sse](#) (int vects, int len, void \*\*array)  
*Checks that array of N sources, P and Q are consistent across all vectors.*
- int [pq\\_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.*

### 7.15.1 Detailed Description

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

### 7.15.2 Function Documentation

#### 7.15.2.1 int [pq\\_check](#) ( int vects, int len, void \*\* array )

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

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

#### Parameters

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

**Returns**

0 pass, other fail

**7.15.2.2 int pq\_check\_base ( int *vects*, int *len*, void \*\* *array* )**

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

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.3 int pq\_check\_sse ( int *vects*, int *len*, void \*\* *array* )**

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

**Requires** SSE4.1

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.4 int pq\_gen ( int *vects*, int *len*, void \*\* *array* )**

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

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

---



**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.5 int pq\_gen\_avx ( int *vects*, int *len*, void \*\* *array* )**

Generate P+Q parity vectors from N sources.

**Requires** AVX

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.6 int pq\_gen\_avx2 ( int *vects*, int *len*, void \*\* *array* )**

Generate P+Q parity vectors from N sources.

**Requires** AVX2

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.7 int pq\_gen\_base ( int *vects*, int *len*, void \*\* *array* )**

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

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.8 int pq\_gen\_sse ( int *vects*, int *len*, void \*\* *array* )**

Generate P+Q parity vectors from N sources.

**Requires** SSE4.1

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.9 int xor\_check ( int *vects*, int *len*, void \*\* *array* )**

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

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

---

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.10 int xor\_check\_base ( int *vects*, int *len*, void \*\* *array* )**

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

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.11 int xor\_check\_sse ( int *vects*, int *len*, void \*\* *array* )**

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

**Requires** SSE4.1

**Parameters**

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

**Returns**

0 pass, other fail

**Examples:**

[xor\\_example.c](#).

---

**7.15.2.12 int xor\_gen ( int *vects*, int *len*, void \*\* *array* )**

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

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

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.13 int xor\_gen\_avx ( int *vects*, int *len*, void \*\* *array* )**

Generate XOR parity vector from N sources.

**Requires** AVX

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.14 int xor\_gen\_base ( int *vects*, int *len*, void \*\* *array* )**

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

**Parameters**

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

**Returns**

0 pass, other fail

**7.15.2.15 int xor\_gen\_sse ( int vects, int len, void \*\* array )**

Generate XOR parity vector from N sources.

**Requires** SSE4.1

**Parameters**

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

**Returns**

0 pass, other fail

**Examples:**

[xor\\_example.c](#).

## 7.16 sha.h File Reference

SHA1 functions.

**Functions**

- void [sha1\\_update](#) (unsigned int \*digest, unsigned char \*input, size\_t num\_blocks)  
*Part of the SHA1 hash algorithm that can be run repeatedly on message blocks of 64 bytes to update the hash value.*
- void [sha1\\_opt](#) (unsigned char \*input, unsigned int \*digest, int len)  
*Performs complete SHA1 algorithm using optimized sha1\_update routine.*

### 7.16.1 Detailed Description

SHA1 functions.

---

## 7.16.2 Function Documentation

### 7.16.2.1 void sha1\_opt ( unsigned char \* *input*, unsigned int \* *digest*, int *len* )

Performs complete SHA1 algorithm using optimized sha1\_update routine.

**Requires** SSE3

#### Parameters

<i>input</i>	Pointer to buffer containing the input message.
<i>digest</i>	Pointer to digest to update.
<i>len</i>	Length of buffer.

#### Returns

None

### 7.16.2.2 void sha1\_update ( unsigned int \* *digest*, unsigned char \* *input*, size\_t *num\_blocks* )

Part of the SHA1 hash algorithm that can be run repeatedly on message blocks of 64 bytes to update the hash value.

**Requires** SSE3

#### Parameters

<i>digest</i>	Pointer to digest to update.
<i>input</i>	Pointer to buffer containing the input message in 64 byte blocks.
<i>num_blocks</i>	Number of 64 byte blocks to incorporate in hash update.

#### Returns

None

## 7.17 sha1\_mb.h File Reference

Multi-buffer CTX API SHA1 function prototypes and structures.

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

---

## Data Structures

- struct [SHA1\\_JOB](#)  
*Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.*
- struct [SHA1\\_MB\\_ARGS\\_X8](#)  
*Scheduler layer - Holds arguments for submitted SHA1 job.*
- struct [SHA1\\_LANE\\_DATA](#)  
*Scheduler layer - Lane data.*
- struct [SHA1\\_MB\\_JOB\\_MGR](#)  
*Scheduler layer - Holds state for multi-buffer SHA1 jobs.*
- struct [SHA1\\_HASH\\_CTX\\_MGR](#)  
*Context layer - Holds state for multi-buffer SHA1 jobs.*
- struct [SHA1\\_HASH\\_CTX](#)  
*Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.*

## Functions

- void [sha1\\_ctx\\_mgr\\_init\\_sse](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the context level SHA1 multi-buffer manager structure.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_submit\\_sse](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA1\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA1 job to the context level multi-buffer manager.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_flush\\_sse](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA1 jobs and return when complete.*
  - void [sha1\\_ctx\\_mgr\\_init\\_avx](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA1 multi-buffer manager structure.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_submit\\_avx](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA1\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA1 job to the multi-buffer manager.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_flush\\_avx](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA1 jobs and return when complete.*
  - void [sha1\\_ctx\\_mgr\\_init\\_avx2](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA1 multi-buffer manager structure.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_submit\\_avx2](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA1\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA1 job to the multi-buffer manager.*
  - [SHA1\\_HASH\\_CTX](#) \* [sha1\\_ctx\\_mgr\\_flush\\_avx2](#) ([SHA1\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA1 jobs and return when complete.*
-

- void `sha1_ctx_mgr_init` (`SHA1_HASH_CTX_MGR *mgr`)  
*Initialize the SHA1 multi-buffer manager structure.*
- `SHA1_HASH_CTX *sha1_ctx_mgr_submit` (`SHA1_HASH_CTX_MGR *mgr`, `SHA1_HASH_CTX *ctx`, const void `*buffer`, `uint32_t len`, `HASH_CTX_FLAG` flags)  
*Submit a new SHA1 job to the multi-buffer manager.*
- `SHA1_HASH_CTX *sha1_ctx_mgr_flush` (`SHA1_HASH_CTX_MGR *mgr`)  
*Finish all submitted SHA1 jobs and return when complete.*

### 7.17.1 Detailed Description

Multi-buffer CTX API SHA1 function prototypes and structures. Interface for multi-buffer SHA1 functions

#### Multi-buffer SHA1 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the `SHA1_HASH_CTX_MGR` and `SHA1_HASH_CTX` objects. Numerous `SHA1_HASH_CTX` objects may be instantiated by the application for use with a single `SHA1_HASH_CTX_MGR`.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the `HASH_ENTIRE` flag, or as partial jobs which can be started using the `HASH_FIRST` flag, and later resumed or finished using the `HASH_UPDATE` and `HASH_LAST` flags respectively.

**Note:** The submit function does not require data buffers to be block sized.

The SHA1 CTX interface functions are available for 3 architectures: SSE, AVX and AVX2. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

**Usage:** The application creates a `SHA1_HASH_CTX_MGR` object and initializes it with a call to `sha1_ctx_mgr_init*` function, where henceforth "\*" stands for the relevant suffix for each architecture; `_sse`, `_avx`, `_avx2` (or no suffix for the multibinary version). The `SHA1_HASH_CTX_MGR` object will be used to schedule processor resources, with up to 4 `SHA1_HASH_CTX` objects (or 8 in the AVX2 case) being processed at a time.

Each `SHA1_HASH_CTX` must be initialized before first use by the `hash_ctx_init` macro defined in `multi_buffer.h`. After initialization, the application may begin computing a hash by giving the `SHA1_HASH_CTX` to a `SHA1_HASH_CTX_MGR` using the submit functions `sha1_ctx_mgr_submit*`() with the `HASH_FIRST` flag set. When the `SHA1_HASH_CTX` is returned to the application (via this or a later call to `sha1_ctx_mgr_submit*`() or `sha1_ctx_mgr_flush*`()), the application can then re-submit it with another call to `sha1_ctx_mgr_submit*`(), but without the `HASH_FIRST` flag set.

Ideally, on the last buffer for that hash, `sha1_ctx_mgr_submit_sse` is called with `HASH_LAST`, although it is also possible to submit the hash with `HASH_LAST` and a zero length if necessary. When a `SHA1_HASH_CTX` is returned after having been submitted with `HASH_LAST`, it will contain a valid hash. The `SHA1_HASH_CTX` can be reused immediately by submitting with `HASH_FIRST`.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: `HASH_FIRST` | `HASH_LAST` (or, equivalently, `HASH_ENTIRE`)



- two buffers: HASH\_FIRST, HASH\_LAST
- three buffers: HASH\_FIRST, HASH\_UPDATE, HASH\_LAST etc.

The order in which SHA1\_CTX objects are returned is in general different from the order in which they are submitted. A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a [SHA1\\_HASH\\_CTX\\_MGR](#).
- Submitting a context after HASH\_LAST is used but before HASH\_FIRST is set.

These error conditions are reported by returning the [SHA1\\_HASH\\_CTX](#) immediately after a submit with its error member set to a non-zero error code (defined in [multi\\_buffer.h](#)). No changes are made to the [SHA1\\_HASH\\_CTX\\_MGR](#) in the case of an error; no processing is done for other hashes.

## 7.17.2 Function Documentation

### 7.17.2.1 SHA1\_HASH\_CTX\* sha1\_ctx\_mgr.flush ( SHA1\_HASH\_CTX\_MGR \* mgr )

Finish all submitted SHA1 jobs and return when complete.

**Requires** SSE4.1 or AVX or AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
------------	--

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.17.2.2 SHA1\_HASH\_CTX\* sha1\_ctx\_mgr.flush\_avx ( SHA1\_HASH\_CTX\_MGR \* mgr )

Finish all submitted SHA1 jobs and return when complete.

**Requires** AVX

#### Parameters

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.17.2.3 SHA1\_HASH\_CTX\* sha1\_ctx\_mgr.flush\_avx2 ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA1 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.17.2.4 SHA1\_HASH\_CTX\* sha1\_ctx\_mgr.flush\_sse ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA1 jobs and return when complete.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.17.2.5 void sha1\_ctx\_mgr\_init ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA1 multi-buffer manager structure.

**Requires** SSE4.1 or AVX or AVX2

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.17.2.6 void sha1\_ctx\_mgr\_init\_avx ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA1 multi-buffer manager structure.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.17.2.7 void sha1\_ctx\_mgr\_init\_avx2 ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA1 multi-buffer manager structure.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.17.2.8 void sha1\_ctx\_mgr\_init\_sse ( SHA1\_HASH\_CTX\_MGR \* mgr )**

Initialize the context level SHA1 multi-buffer manager structure.

**Requires** SSE4.1

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.17.2.9** `SHA1_HASH_CTX* sha1_ctx_mgr_submit ( SHA1_HASH_CTX_MGR * mgr, SHA1_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA1 job to the multi-buffer manager.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.17.2.10** `SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx ( SHA1_HASH_CTX_MGR * mgr, SHA1_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA1 job to the multi-buffer manager.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.17.2.11** `SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx2 ( SHA1_HASH_CTX_MGR * mgr,  
SHA1_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA1 job to the multi-buffer manager.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer (in bytes) to be processed
<i>len</i>	Length of buffer to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.17.2.12** `SHA1_HASH_CTX* sha1_ctx_mgr_submit_sse ( SHA1_HASH_CTX_MGR * mgr,  
SHA1_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA1 job to the context level multi-buffer manager.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

---

## 7.18 sha256\_mb.h File Reference

Multi-buffer CTX API SHA256 function prototypes and structures.

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

### Data Structures

- struct [SHA256\\_JOB](#)  
*Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.*
- struct [SHA256\\_MB\\_ARGS\\_X8](#)  
*Scheduler layer - Holds arguments for submitted SHA256 job.*
- struct [SHA256\\_LANE\\_DATA](#)  
*Scheduler layer - Lane data.*
- struct [SHA256\\_MB\\_JOB\\_MGR](#)  
*Scheduler layer - Holds state for multi-buffer SHA256 jobs.*
- struct [SHA256\\_HASH\\_CTX\\_MGR](#)  
*Context layer - Holds state for multi-buffer SHA256 jobs.*
- struct [SHA256\\_HASH\\_CTX](#)  
*Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.*

### Functions

- void [sha256\\_ctx\\_mgr\\_init\\_sse](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the context level SHA256 multi-buffer manager structure.*
  - [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_submit\\_sse](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA256\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA256 job to the context level multi-buffer manager.*
  - [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_flush\\_sse](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA256 jobs and return when complete.*
  - void [sha256\\_ctx\\_mgr\\_init\\_avx](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA256 multi-buffer manager structure.*
  - [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_submit\\_avx](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA256\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA256 job to the multi-buffer manager.*
  - [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_flush\\_avx](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA256 jobs and return when complete.*
-

- void [sha256\\_ctx\\_mgr\\_init\\_avx2](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA256 multi-buffer manager structure.*
- [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_submit\\_avx2](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA256\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA256 job to the multi-buffer manager.*
- [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_flush\\_avx2](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA256 jobs and return when complete.*
- void [sha256\\_ctx\\_mgr\\_init](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA256 multi-buffer manager structure.*
- [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_submit](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA256\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA256 job to the multi-buffer manager.*
- [SHA256\\_HASH\\_CTX](#) \* [sha256\\_ctx\\_mgr\\_flush](#) ([SHA256\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA256 jobs and return when complete.*

### 7.18.1 Detailed Description

Multi-buffer CTX API SHA256 function prototypes and structures. Interface for multi-buffer SHA256 functions

#### Multi-buffer SHA256 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the [SHA256\\_HASH\\_CTX\\_MGR](#) and [SHA256\\_HASH\\_CTX](#) objects. Numerous [SHA256\\_HASH\\_CTX](#) objects may be instantiated by the application for use with a single [SHA256\\_HASH\\_CTX\\_MGR](#).

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the [HASH\\_ENTIRE](#) flag, or as partial jobs which can be started using the [HASH\\_FIRST](#) flag, and later resumed or finished using the [HASH\\_UPDATE](#) and [HASH\\_LAST](#) flags respectively.

**Note:** The submit function does not require data buffers to be block sized.

The SHA256 CTX interface functions are available for 3 architectures: SSE, AVX and AVX2. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

**Usage:** The application creates a [SHA256\\_HASH\\_CTX\\_MGR](#) object and initializes it with a call to [sha256\\_ctx\\_mgr\\_init\\*](#)() function, where henceforth "\*" stands for the relevant suffix for each architecture; [\\_sse](#), [\\_avx](#), [\\_avx2](#) (or no suffix for the multibinary version). The [SHA256\\_HASH\\_CTX\\_MGR](#) object will be used to schedule processor resources, with up to 4 [SHA256\\_HASH\\_CTX](#) objects (or 8 in the AVX2 case) being processed at a time.

Each [SHA256\\_HASH\\_CTX](#) must be initialized before first use by the [hash\\_ctx\\_init](#) macro defined in [multi\\_buffer.h](#). After initialization, the application may begin computing a hash by giving the [SHA256\\_HASH\\_CTX](#) to a [SHA256\\_HASH\\_CTX\\_MGR](#) using the submit functions [sha256\\_ctx\\_mgr\\_submit\\*](#)() with the [HASH\\_FIRST](#) flag set. When the [SHA256\\_HASH\\_CTX](#) is returned to the application (via this or a later call to [sha256\\_ctx\\_mgr\\_submit\\*](#)() or [sha256\\_ctx\\_mgr\\_flush\\*](#)()), the application can then re-submit it with another call to [sha256\\_ctx\\_mgr\\_submit\\*](#)(), but without the [HASH\\_FIRST](#) flag set.

Ideally, on the last buffer for that hash, `sha256_ctx_mgr_submit_sse` is called with `HASH_LAST`, although it is also possible to submit the hash with `HASH_LAST` and a zero length if necessary. When a [SHA256\\_HASH\\_CTX](#) is returned after having been submitted with `HASH_LAST`, it will contain a valid hash. The [SHA256\\_HASH\\_CTX](#) can be reused immediately by submitting with `HASH_FIRST`.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: `HASH_FIRST | HASH_LAST` (or, equivalently, `HASH_ENTIRE`)
- two buffers: `HASH_FIRST`, `HASH_LAST`
- three buffers: `HASH_FIRST`, `HASH_UPDATE`, `HASH_LAST` etc.

The order in which `SHA256_CTX` objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a [SHA256\\_HASH\\_CTX\\_MGR](#).
- Submitting a context after `HASH_LAST` is used but before `HASH_FIRST` is set.

These error conditions are reported by returning the [SHA256\\_HASH\\_CTX](#) immediately after a submit with its error member set to a non-zero error code (defined in [multi\\_buffer.h](#)). No changes are made to the [SHA256\\_HASH\\_CTX\\_MGR](#) in the case of an error; no processing is done for other hashes.

## 7.18.2 Function Documentation

### 7.18.2.1 `SHA256_HASH_CTX* sha256_ctx_mgr_flush ( SHA256_HASH_CTX_MGR * mgr )`

Finish all submitted SHA256 jobs and return when complete.

**Requires** SSE4.1 or AVX or AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
------------	--

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

---



**7.18.2.2 SHA256\_HASH\_CTX\* sha256\_ctx\_mgr\_flush\_avx ( SHA256\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA256 jobs and return when complete.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.18.2.3 SHA256\_HASH\_CTX\* sha256\_ctx\_mgr\_flush\_avx2 ( SHA256\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA256 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.18.2.4 SHA256\_HASH\_CTX\* sha256\_ctx\_mgr\_flush\_sse ( SHA256\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA256 jobs and return when complete.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

---

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.18.2.5 void sha256\_ctx\_mgr\_init ( SHA256\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the SHA256 multi-buffer manager structure.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.18.2.6 void sha256\_ctx\_mgr\_init\_avx ( SHA256\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the SHA256 multi-buffer manager structure.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.18.2.7 void sha256\_ctx\_mgr\_init\_avx2 ( SHA256\_HASH\_CTX\_MGR \* *mgr* )**

Initialize the SHA256 multi-buffer manager structure.

**Requires** AVX2

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.18.2.8 void sha256\_ctx\_mgr\_init\_sse ( SHA256\_HASH\_CTX\_MGR \* mgr )**

Initialize the context level SHA256 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.18.2.9 SHA256\_HASH\_CTX\* sha256\_ctx\_mgr\_submit ( SHA256\_HASH\_CTX\_MGR \* mgr, SHA256\_HASH\_CTX \* ctx, const void \* buffer, uint32\_t len, HASH\_CTX\_FLAG flags )**

Submit a new SHA256 job to the multi-buffer manager.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.18.2.10** `SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx ( SHA256_HASH_CTX_MGR * mgr, SHA256_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA256 job to the multi-buffer manager.

**Requires** AVX

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.18.2.11** `SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx2 ( SHA256_HASH_CTX_MGR * mgr, SHA256_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA256 job to the multi-buffer manager.

**Requires** AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.18.2.12** `SHA256_HASH_CTX* sha256_ctx_mgr_submit_sse ( SHA256_HASH_CTX_MGR * mgr, SHA256_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA256 job to the context level multi-buffer manager.

---

**Requires** SSE4.1

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

## 7.19 sha512\_mb.h File Reference

Single/Multi-buffer CTX API SHA512 function prototypes and structures.

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

#### Data Structures

- struct [SHA512\\_JOB](#)  
*Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.*
  - struct [SHA512\\_MB\\_ARGS\\_X4](#)  
*Scheduler layer - Holds arguments for submitted SHA512 job.*
  - struct [SHA512\\_LANE\\_DATA](#)  
*Scheduler layer - Lane data.*
  - struct [SHA512\\_MB\\_JOB\\_MGR](#)  
*Scheduler layer - Holds state for multi-buffer SHA512 jobs.*
  - struct [SHA512\\_HASH\\_CTX\\_MGR](#)  
*Context layer - Holds state for multi-buffer SHA512 jobs.*
  - struct [SHA512\\_HASH\\_CTX](#)  
*Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.*
-

## Functions

- void [sha512\\_ctx\\_mgr\\_init\\_sse](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the context level SHA512 multi-buffer manager structure.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_submit\\_sse](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA512\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA512 job to the context level multi-buffer manager.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_flush\\_sse](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA512 jobs and return when complete.*
  - void [sha512\\_ctx\\_mgr\\_init\\_avx](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA512 multi-buffer manager structure.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_submit\\_avx](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA512\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_flush\\_avx](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA512 jobs and return when complete.*
  - void [sha512\\_ctx\\_mgr\\_init\\_avx2](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA512 multi-buffer manager structure.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_submit\\_avx2](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA512\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_flush\\_avx2](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA512 jobs and return when complete.*
  - void [sha512\\_ctx\\_mgr\\_init\\_sb\\_sse4](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA512 multi-buffer manager structure.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_submit\\_sb\\_sse4](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA512\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_flush\\_sb\\_sse4](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA512 jobs and return when complete.*
  - void [sha512\\_ctx\\_mgr\\_init](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Initialize the SHA512 multi-buffer manager structure.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_submit](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr, [SHA512\\_HASH\\_CTX](#) \*ctx, const void \*buffer, uint32\_t len, [HASH\\_CTX\\_FLAG](#) flags)  
*Submit a new SHA512 job to the multi-buffer manager.*
  - [SHA512\\_HASH\\_CTX](#) \* [sha512\\_ctx\\_mgr\\_flush](#) ([SHA512\\_HASH\\_CTX\\_MGR](#) \*mgr)  
*Finish all submitted SHA512 jobs and return when complete.*
-

### 7.19.1 Detailed Description

Single/Multi-buffer CTX API SHA512 function prototypes and structures. Interface for single and multi-buffer SHA512 functions

#### Single/Multi-buffer SHA512 Entire or First-Update..Update-Last

The interface to this single/multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the [SHA512\\_HASH\\_CTX\\_MGR](#) and [SHA512\\_HASH\\_CTX](#) objects. Numerous [SHA512\\_HASH\\_CTX](#) objects may be instantiated by the application for use with a single [SHA512\\_HASH\\_CTX\\_MGR](#).

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the HASH\_ENTIRE flag, or as partial jobs which can be started using the HASH\_FIRST flag, and later resumed or finished using the HASH\_UPDATE and HASH\_LAST flags respectively.

**Note:** The submit function does not require data buffers to be block sized.

The SHA512 CTX interface functions are available for 4 architectures: multi-buffer SSE, AVX and AVX2, and single-buffer SSE4 (which is used in the same way as the multi-buffer code). In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime. This multibinary interface selects the single buffer SSE4 functions when the platform is detected to be Silvermont.

**Usage:** The application creates a [SHA512\\_HASH\\_CTX\\_MGR](#) object and initializes it with a call to `sha512_ctx_mgr_init*()` function, where henceforth "\*" stands for the relevant suffix for each architecture; `_sse`, `_avx`, `_avx2` (or no suffix for the multibinary version). The [SHA512\\_HASH\\_CTX\\_MGR](#) object will be used to schedule processor resources, with up to 4 [SHA512\\_HASH\\_CTX](#) objects (or 8 in the AVX2 case) being processed at a time.

Each [SHA512\\_HASH\\_CTX](#) must be initialized before first use by the `hash_ctx_init` macro defined in [multi\\_buffer.h](#). After initialization, the application may begin computing a hash by giving the [SHA512\\_HASH\\_CTX](#) to a [SHA512\\_HASH\\_CTX\\_MGR](#) using the submit functions `sha512_ctx_mgr_submit*()` with the HASH\_FIRST flag set. When the [SHA512\\_HASH\\_CTX](#) is returned to the application (via this or a later call to `sha512_ctx_mgr_submit*()` or `sha512_ctx_mgr_flush*()`), the application can then re-submit it with another call to `sha512_ctx_mgr_submit*()`, but without the HASH\_FIRST flag set.

Ideally, on the last buffer for that hash, `sha512_ctx_mgr_submit_sse` is called with HASH\_LAST, although it is also possible to submit the hash with HASH\_LAST and a zero length if necessary. When a [SHA512\\_HASH\\_CTX](#) is returned after having been submitted with HASH\_LAST, it will contain a valid hash. The [SHA512\\_HASH\\_CTX](#) can be reused immediately by submitting with HASH\_FIRST.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: HASH\_FIRST | HASH\_LAST (or, equivalently, HASH\_ENTIRE)
- two buffers: HASH\_FIRST, HASH\_LAST
- three buffers: HASH\_FIRST, HASH\_UPDATE, HASH\_LAST etc.

The order in which SHA512\_CTX objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

---

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a [SHA512\\_HASH\\_CTX\\_MGR](#). (Note: This error case is not applicable to the single buffer SSE4 version)
- Submitting a context after HASH\_LAST is used but before HASH\_FIRST is set.

These error conditions are reported by returning the [SHA512\\_HASH\\_CTX](#) immediately after a submit with its error member set to a non-zero error code (defined in [multi\\_buffer.h](#)). No changes are made to the [SHA512\\_HASH\\_CTX\\_MGR](#) in the case of an error; no processing is done for other hashes.

## 7.19.2 Function Documentation

### 7.19.2.1 [SHA512\\_HASH\\_CTX](#)\* [sha512\\_ctx\\_mgr\\_flush](#) ( [SHA512\\_HASH\\_CTX\\_MGR](#) \* *mgr* )

Finish all submitted SHA512 jobs and return when complete.

**Requires** SSE4.1 or AVX or AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
------------	--

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

### 7.19.2.2 [SHA512\\_HASH\\_CTX](#)\* [sha512\\_ctx\\_mgr\\_flush\\_avx](#) ( [SHA512\\_HASH\\_CTX\\_MGR](#) \* *mgr* )

Finish all submitted SHA512 jobs and return when complete.

**Requires** AVX

#### Parameters

<i>mgr</i>	Structure holding context level state info
------------	--

#### Returns

NULL if no jobs to complete or pointer to jobs structure.

---



**7.19.2.3 SHA512\_HASH\_CTX\* sha512\_ctx\_mgr\_flush\_avx2 ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA512 jobs and return when complete.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.19.2.4 SHA512\_HASH\_CTX\* sha512\_ctx\_mgr\_flush\_sb\_sse4 ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA512 jobs and return when complete.

**Requires** SSE4

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.19.2.5 SHA512\_HASH\_CTX\* sha512\_ctx\_mgr\_flush\_sse ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Finish all submitted SHA512 jobs and return when complete.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

---

**Returns**

NULL if no jobs to complete or pointer to jobs structure.

**7.19.2.6 void sha512\_ctx\_mgr\_init ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** SSE4.1 or AVX or AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.19.2.7 void sha512\_ctx\_mgr\_init\_avx ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** AVX

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.19.2.8 void sha512\_ctx\_mgr\_init\_avx2 ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** AVX2

---

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.19.2.9 void sha512\_ctx\_mgr\_init\_sb\_sse4 ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Initialize the SHA512 multi-buffer manager structure.

**Requires** SSE4

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.19.2.10 void sha512\_ctx\_mgr\_init\_sse ( SHA512\_HASH\_CTX\_MGR \* mgr )**

Initialize the context level SHA512 multi-buffer manager structure.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
------------	--

**Returns**

void

**7.19.2.11 SHA512\_HASH\_CTX\* sha512\_ctx\_mgr\_submit ( SHA512\_HASH\_CTX\_MGR \* mgr, SHA512\_HASH\_CTX \* ctx, const void \* buffer, uint32\_t len, HASH\_CTX\_FLAG flags )**

Submit a new SHA512 job to the multi-buffer manager.

---

**Requires** SSE4.1 or AVX or AVX2

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.19.2.12** `SHA512_HASH_CTX* sha512_ctx_mgr_submit_avx ( SHA512_HASH_CTX_MGR * mgr, SHA512_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA512 job to the multi-buffer manager.

**Requires** AVX

#### Parameters

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

#### Returns

NULL if no jobs complete or pointer to jobs structure.

**7.19.2.13** `SHA512_HASH_CTX* sha512_ctx_mgr_submit_avx2 ( SHA512_HASH_CTX_MGR * mgr, SHA512_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA512 job to the multi-buffer manager.

**Requires** AVX2

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.19.2.14** `SHA512_HASH_CTX* sha512_ctx_mgr_submit_sb_sse4 ( SHA512_HASH_CTX_MGR * mgr, SHA512_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA512 job to the multi-buffer manager.

**Requires** SSE4

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

**7.19.2.15** `SHA512_HASH_CTX* sha512_ctx_mgr_submit_sse ( SHA512_HASH_CTX_MGR * mgr, SHA512_HASH_CTX * ctx, const void * buffer, uint32_t len, HASH_CTX_FLAG flags )`

Submit a new SHA512 job to the context level multi-buffer manager.

**Requires** SSE4.1

**Parameters**

<i>mgr</i>	Structure holding context level state info
<i>ctx</i>	Structure holding ctx job info
<i>buffer</i>	Pointer to buffer to be processed
<i>len</i>	Length of buffer (in bytes) to be processed
<i>flags</i>	Input flag specifying job type (first, update, last or entire)

**Returns**

NULL if no jobs complete or pointer to jobs structure.

## 7.20 types.h File Reference

Defines standard width types.

### 7.20.1 Detailed Description

Defines standard width types.

---

# CHAPTER 8

## EXAMPLE DOCUMENTATION

---

### 8.1 crc\_simple\_test.c

Example usage of crc multibinary functions.

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

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

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

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

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

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

    t10_dif_computed = crc16_t10dif(init_crc_16, p_buf, 48);

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

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

```

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

    return 0;
}

```

## 8.2 igzip\_example.c

Example simple application using fast\_lz.

```

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

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

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

#define BUF_SIZE 8192

LZ_Stream1 stream;

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

    if (argc != 3) {
        fprintf(stderr, "Usage: igzip_example infile outfile\n");
        exit(0);
    }
    in = fopen(argv[1], "rb");
    if (!in) {
        fprintf(stderr, "Can't open %s for reading\n", argv[1]);
        exit(0);
    }
}

```



```

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

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

init_stream(&stream);
stream.end_of_stream = 0;

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

        fast_lz(&stream);

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

    assert(stream.avail_in == 0);
} while (!stream.end_of_stream);

fclose(out);
fclose(in);

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

```

## 8.3 multi\_buffer\_sha1\_example.c

Example of multi-buffer hashing using mb\_sha1.

```

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

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

```

```

THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*****/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "mb_shal.h"

// Test messages
#define TST_STR "0123456789;<=>?@ABCDEFGHIJKLMNQRSTUUVWX"
uint8_t msg1[] = "abcdcbcdcedfdefgefghfghighijhijkijklklmklmnlmnomnopnopq";
uint8_t msg2[] = "0123456789;<=>?@ABCDEFGHIJKLMNO";
uint8_t msg3[] = TST_STR TST_STR "0123456789;<";
uint8_t msg4[] = TST_STR TST_STR TST_STR "0123456789;<=>?@ABCDEFGHIJKLMNQRST";
uint8_t msg5[] = TST_STR TST_STR TST_STR TST_STR TST_STR "0123456789;<=>?";
uint8_t msg6[] =
    TST_STR TST_STR TST_STR TST_STR TST_STR TST_STR "0123456789;<=>?@ABCDEFGHIJKLMNQRSTU";
uint8_t msg7[] = "";

// Expected digests
uint32_t dgst1[] = { 0x84983E44, 0x1C3BD26E, 0xBAAE4AA1, 0xF95129E5, 0xE54670F1 };
uint32_t dgst2[] = { 0xB7C66452, 0x0FD122B3, 0x55D539F2, 0xA35E6FAA, 0xC2A5A11D };
uint32_t dgst3[] = { 0x127729B6, 0xA8B2F8A0, 0xA4DDC819, 0x08E1D8B3, 0x67CEEAA55 };
uint32_t dgst4[] = { 0xFDDE2D00, 0xABD5B7A3, 0x699DE6F2, 0x3FF1D1AC, 0x3B872AC2 };
uint32_t dgst5[] = { 0xE7FCA85C, 0xA4AB3740, 0x6A180B32, 0x0B8D362C, 0x622A96E6 };
uint32_t dgst6[] = { 0x505B0686, 0xE1ACDF42, 0xB3588B5A, 0xB043D52C, 0x6D8C7444 };
uint32_t dgst7[] = { 0xDA39A3EE, 0x5E6B4B0D, 0x3255BFEF, 0x95601890, 0xAFD80709 };

uint8_t *msgs[] = { msg1, msg2, msg3, msg4, msg5, msg6, msg7 };
uint32_t *expected_digest[] = { dgst1, dgst2, dgst3, dgst4, dgst5, dgst6, dgst7 };

int check_job(uint32_t * ref, uint32_t * good, int words)
{
    int i;
    for (i = 0; i < words; i++)
        if (good[i] != ref[i])
            return 1;

    return 0;
}

#define MAX_MSGS 7

int main()
{
    SHA1_MB_MGR mb_mgr;
    JOB_SHA1 job[MAX_MSGS], *p_job;
    int i, checked = 0, failed = 0;
    int n = sizeof(msgs) / sizeof(msgs[0]);

    // Set up jobs
    for (i = 0; i < n; i++) {
        job[i].buffer = msgs[i];
        job[i].len = strlen((char *)msgs[i]);
        job[i].len_total = job[i].len;
        job[i].user_data = (void *)expected_digest[i];
        job[i].flags = HASH_MB_FIRST | HASH_MB_LAST;
    }

    // Initialize multi-buffer manager
    shal_init_mb_mgr(&mb_mgr);

    for (i = 0; i < n; i++) {
        p_job = shal_submit_job(&mb_mgr, &job[i]);

        if (p_job) { // If we have finished a job, process it
            checked++;
            failed +=

```

```

        check_job(p_job->result_digest, p_job->
user_data,
                NUM_SHA1_DIGEST_WORDS);
    }

    // Finish remaining jobs
    while (NULL != (p_job = sha1_flush_job(&mb_mgr))) {
        checked++;
        failed +=
            check_job(p_job->result_digest, p_job->
user_data, NUM_SHA1_DIGEST_WORDS);
    }

    printf("Example multi-buffer sha1 completed=%d, failed=%d\n", checked, failed);
    return 0;
}

```

## 8.4 xor\_example.c

Example of XOR usage on multiple sources.

```

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

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

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

#define TEST_SOURCES 16
#define TEST_LEN    16*1024

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

    printf("XOR example\n");

```

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

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

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

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

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

return 0;
}
```

## INDEX

aes\_keyexp\_128  
    aes\_xts.h, 58  
aes\_keyexp\_256  
    aes\_xts.h, 58  
aes\_xts.h, 56  
    aes\_keyexp\_128, 58  
    aes\_keyexp\_256, 58  
    XTS\_AES\_128\_dec, 58  
    XTS\_AES\_128\_dec\_expanded\_key, 59  
    XTS\_AES\_128\_enc, 59  
    XTS\_AES\_128\_enc\_expanded\_key, 59  
    XTS\_AES\_256\_dec, 60  
    XTS\_AES\_256\_dec\_expanded\_key, 60  
    XTS\_AES\_256\_enc, 61  
    XTS\_AES\_256\_enc\_expanded\_key, 61  
BitBuf2, 26  
crc.h, 62  
    crc16\_t10dif, 63  
    crc16\_t10dif\_01, 63  
    crc16\_t10dif\_base, 63  
    crc16\_t10dif\_by4, 64  
    crc32\_ieee, 64  
    crc32\_ieee\_01, 65  
    crc32\_ieee\_base, 65  
    crc32\_ieee\_by4, 65  
    crc32\_iscsi, 66  
    crc32\_iscsi\_00, 66  
    crc32\_iscsi\_01, 66  
    crc32\_iscsi\_base, 67  
    crc32\_iscsi\_baseline, 67  
    crc32\_iscsi\_simple, 68  
crc16\_t10dif  
    crc.h, 63  
crc16\_t10dif\_01  
    crc.h, 63  
crc16\_t10dif\_base  
    crc.h, 63  
crc16\_t10dif\_by4  
    crc.h, 64  
crc32\_ieee  
    crc.h, 64  
crc32\_ieee\_01  
    crc.h, 65  
crc32\_ieee\_base  
    crc.h, 65  
crc32\_ieee\_by4  
    crc.h, 65  
crc32\_iscsi  
    crc.h, 66  
crc32\_iscsi\_00  
    crc.h, 66  
crc32\_iscsi\_01  
    crc.h, 66  
crc32\_iscsi\_base  
    crc.h, 67  
crc32\_iscsi\_baseline  
    crc.h, 67  
crc32\_iscsi\_simple  
    crc.h, 68  
ec\_encode\_data  
    erasure\_code.h, 72  
ec\_encode\_data\_avx  
    erasure\_code.h, 73  
ec\_encode\_data\_avx2  
    erasure\_code.h, 73  
ec\_encode\_data\_base  
    erasure\_code.h, 73  
ec\_encode\_data\_sse  
    erasure\_code.h, 73  
ec\_encode\_data\_update  
    erasure\_code.h, 74  
ec\_encode\_data\_update\_avx  
    erasure\_code.h, 74  
ec\_encode\_data\_update\_avx2  
    erasure\_code.h, 74  
ec\_encode\_data\_update\_base  
    erasure\_code.h, 75  
ec\_encode\_data\_update\_sse  
    erasure\_code.h, 75  
ec\_init\_tables  
    erasure\_code.h, 75  
erasure\_code.h, 68  
    ec\_encode\_data, 72  
    ec\_encode\_data\_avx, 73  
    ec\_encode\_data\_avx2, 73

---

ec\_encode\_data\_base, 73  
ec\_encode\_data\_sse, 73  
ec\_encode\_data\_update, 74  
ec\_encode\_data\_update\_avx, 74  
ec\_encode\_data\_update\_avx2, 74  
ec\_encode\_data\_update\_base, 75  
ec\_encode\_data\_update\_sse, 75  
ec\_init\_tables, 75  
gf\_2vect\_dot\_prod\_avx, 75  
gf\_2vect\_dot\_prod\_avx2, 76  
gf\_2vect\_dot\_prod\_sse, 77  
gf\_2vect\_mad\_avx, 77  
gf\_2vect\_mad\_avx2, 77  
gf\_2vect\_mad\_sse, 77  
gf\_3vect\_dot\_prod\_avx, 78  
gf\_3vect\_dot\_prod\_avx2, 79  
gf\_3vect\_dot\_prod\_sse, 79  
gf\_3vect\_mad\_avx, 80  
gf\_3vect\_mad\_avx2, 80  
gf\_3vect\_mad\_sse, 80  
gf\_4vect\_dot\_prod\_avx, 81  
gf\_4vect\_dot\_prod\_avx2, 81  
gf\_4vect\_dot\_prod\_sse, 82  
gf\_4vect\_mad\_avx, 83  
gf\_4vect\_mad\_avx2, 83  
gf\_4vect\_mad\_sse, 83  
gf\_5vect\_dot\_prod\_avx, 84  
gf\_5vect\_dot\_prod\_avx2, 84  
gf\_5vect\_dot\_prod\_sse, 85  
gf\_5vect\_mad\_avx, 85  
gf\_5vect\_mad\_avx2, 85  
gf\_5vect\_mad\_sse, 86  
gf\_6vect\_dot\_prod\_avx, 86  
gf\_6vect\_dot\_prod\_avx2, 87  
gf\_6vect\_dot\_prod\_sse, 87  
gf\_6vect\_mad\_avx, 88  
gf\_6vect\_mad\_avx2, 88  
gf\_6vect\_mad\_sse, 88  
gf\_gen\_cauchy1\_matrix, 88  
gf\_gen\_rs\_matrix, 89  
gf\_inv, 89  
gf\_invert\_matrix, 89  
gf\_mul, 90  
gf\_vect\_dot\_prod, 90  
gf\_vect\_dot\_prod\_avx, 91  
gf\_vect\_dot\_prod\_avx2, 91  
gf\_vect\_dot\_prod\_base, 92  
gf\_vect\_dot\_prod\_sse, 92  
gf\_vect\_mad, 93  
gf\_vect\_mad\_avx, 93  
gf\_vect\_mad\_avx2, 93  
gf\_vect\_mad\_base, 93  
gf\_vect\_mad\_sse, 94  
fast\_lz  
    igzip\_lib.h, 99  
fast\_lz\_stateless  
    igzip\_lib.h, 100  
gf\_2vect\_dot\_prod\_avx  
    erasure\_code.h, 75  
gf\_2vect\_dot\_prod\_avx2  
    erasure\_code.h, 76  
gf\_2vect\_dot\_prod\_sse  
    erasure\_code.h, 77  
gf\_2vect\_mad\_avx  
    erasure\_code.h, 77  
gf\_2vect\_mad\_avx2  
    erasure\_code.h, 77  
gf\_2vect\_mad\_sse  
    erasure\_code.h, 77  
gf\_3vect\_dot\_prod\_avx  
    erasure\_code.h, 78  
gf\_3vect\_dot\_prod\_avx2  
    erasure\_code.h, 79  
gf\_3vect\_dot\_prod\_sse  
    erasure\_code.h, 79  
gf\_3vect\_mad\_avx  
    erasure\_code.h, 80  
gf\_3vect\_mad\_avx2  
    erasure\_code.h, 80  
gf\_3vect\_mad\_sse  
    erasure\_code.h, 80  
gf\_4vect\_dot\_prod\_avx  
    erasure\_code.h, 81  
gf\_4vect\_dot\_prod\_avx2  
    erasure\_code.h, 81  
gf\_4vect\_dot\_prod\_sse  
    erasure\_code.h, 82  
gf\_4vect\_mad\_avx  
    erasure\_code.h, 83  
gf\_4vect\_mad\_avx2  
    erasure\_code.h, 83

---

gf\_4vect\_mad\_sse  
    erasure\_code.h, 83  
gf\_5vect\_dot\_prod\_avx  
    erasure\_code.h, 84  
gf\_5vect\_dot\_prod\_avx2  
    erasure\_code.h, 84  
gf\_5vect\_dot\_prod\_sse  
    erasure\_code.h, 85  
gf\_5vect\_mad\_avx  
    erasure\_code.h, 85  
gf\_5vect\_mad\_avx2  
    erasure\_code.h, 85  
gf\_5vect\_mad\_sse  
    erasure\_code.h, 86  
gf\_6vect\_dot\_prod\_avx  
    erasure\_code.h, 86  
gf\_6vect\_dot\_prod\_avx2  
    erasure\_code.h, 87  
gf\_6vect\_dot\_prod\_sse  
    erasure\_code.h, 87  
gf\_6vect\_mad\_avx  
    erasure\_code.h, 88  
gf\_6vect\_mad\_avx2  
    erasure\_code.h, 88  
gf\_6vect\_mad\_sse  
    erasure\_code.h, 88  
gf\_gen\_cauchy1\_matrix  
    erasure\_code.h, 88  
gf\_gen\_rs\_matrix  
    erasure\_code.h, 89  
gf\_inv  
    erasure\_code.h, 89  
gf\_invert\_matrix  
    erasure\_code.h, 89  
gf\_mul  
    erasure\_code.h, 90  
gf\_vect\_dot\_prod  
    erasure\_code.h, 90  
gf\_vect\_dot\_prod\_avx  
    erasure\_code.h, 91  
gf\_vect\_dot\_prod\_avx2  
    erasure\_code.h, 91  
gf\_vect\_dot\_prod\_base  
    erasure\_code.h, 92  
gf\_vect\_dot\_prod\_sse  
    erasure\_code.h, 92

gf\_vect\_mad  
    erasure\_code.h, 93  
gf\_vect\_mad\_avx  
    erasure\_code.h, 93  
gf\_vect\_mad\_avx2  
    erasure\_code.h, 93  
gf\_vect\_mad\_base  
    erasure\_code.h, 93  
gf\_vect\_mad\_sse  
    erasure\_code.h, 94  
gf\_vect\_mul  
    gf\_vect\_mul.h, 95  
gf\_vect\_mul.h, 94  
    gf\_vect\_mul, 95  
    gf\_vect\_mul\_avx, 95  
    gf\_vect\_mul\_base, 95  
    gf\_vect\_mul\_init, 96  
    gf\_vect\_mul\_sse, 96  
gf\_vect\_mul\_avx  
    gf\_vect\_mul.h, 95  
gf\_vect\_mul\_base  
    gf\_vect\_mul.h, 95  
gf\_vect\_mul\_init  
    gf\_vect\_mul.h, 96  
gf\_vect\_mul\_sse  
    gf\_vect\_mul.h, 96

HASH\_CTX\_ERROR\_ALREADY\_COMPLETED  
    multi\_buffer.h, 130  
HASH\_CTX\_ERROR\_ALREADY\_PROCESSING  
    multi\_buffer.h, 130  
HASH\_CTX\_ERROR\_INVALID\_FLAGS  
    multi\_buffer.h, 130  
HASH\_CTX\_ERROR\_NONE  
    multi\_buffer.h, 130  
HASH\_CTX\_STS\_COMPLETE  
    multi\_buffer.h, 131  
HASH\_CTX\_STS\_IDLE  
    multi\_buffer.h, 131  
HASH\_CTX\_STS\_LAST  
    multi\_buffer.h, 131  
HASH\_CTX\_STS\_PROCESSING  
    multi\_buffer.h, 131  
HASH\_ENTIRE  
    multi\_buffer.h, 130  
HASH\_FIRST

- multi\_buffer.h, 130
  - HASH\_LAST
    - multi\_buffer.h, 130
  - HASH\_UPDATE
    - multi\_buffer.h, 130
  - HASH\_CTX\_ERROR
    - multi\_buffer.h, 130
  - HASH\_CTX\_FLAG
    - multi\_buffer.h, 130
  - HASH\_CTX\_STS
    - multi\_buffer.h, 130
  - igzip\_lib.h
    - LZS2\_BODY, 99
    - LZS2\_END, 99
    - LZS2\_HDR, 99
    - LZS2\_TRL, 99
  - igzip\_lib.h, 97
    - fast\_lz, 99
    - fast\_lz\_stateless, 100
    - init\_stream, 101
    - LZ\_State1\_state, 99
  - init\_stream
    - igzip\_lib.h, 101
  - intrinreg.h, 101
  - JOB\_MD5, 26
  - JOB\_SHA1, 27
  - JOB\_SHA256, 28
  - JOB\_SHA512, 29
  - JOB\_STS
    - multi\_buffer.h, 131
  - LZS2\_BODY
    - igzip\_lib.h, 99
  - LZS2\_END
    - igzip\_lib.h, 99
  - LZS2\_HDR
    - igzip\_lib.h, 99
  - LZS2\_TRL
    - igzip\_lib.h, 99
  - LZ\_State1, 30
  - LZ\_State1\_state
    - igzip\_lib.h, 99
  - LZ\_Stream1, 31
  - MD5\_ARGS\_X8, 32
  - MD5\_ARGS\_X8X2, 33
  - MD5\_HASH\_CTX, 33
  - MD5\_HASH\_CTX\_MGR, 34
  - MD5\_HMAC\_LANE\_DATA, 34
  - MD5\_JOB, 35
  - MD5\_LANE\_DATA, 35
  - MD5\_MB\_ARGS\_X16, 36
  - MD5\_MB\_JOB\_MGR, 36
  - MD5\_MB\_MGR, 37
  - MD5\_MB\_MGR\_X8X2, 37
  - mb\_md5.h, 101
    - md5\_flush\_job, 103
    - md5\_flush\_job\_avx, 103
    - md5\_flush\_job\_avx2, 103
    - md5\_init\_mb\_mgr, 104
    - md5\_init\_mb\_mgr\_x8x2, 104
    - md5\_submit\_job, 104
    - md5\_submit\_job\_avx, 105
    - md5\_submit\_job\_avx2, 105
  - mb\_sha1.h, 106
    - sha1\_flush\_job, 107
    - sha1\_flush\_job\_avx, 107
    - sha1\_flush\_job\_avx2, 108
    - sha1\_init\_mb\_mgr, 108
    - sha1\_init\_mb\_mgr\_x8, 108
    - sha1\_submit\_job, 109
    - sha1\_submit\_job\_avx, 109
    - sha1\_submit\_job\_avx2, 110
  - mb\_sha256.h, 110
    - sha256\_flush\_job, 112
    - sha256\_flush\_job\_avx, 112
    - sha256\_flush\_job\_avx2, 112
    - sha256\_init\_mb\_mgr, 113
    - sha256\_init\_mb\_mgr\_x8, 113
    - sha256\_submit\_job, 113
    - sha256\_submit\_job\_avx, 114
    - sha256\_submit\_job\_avx2, 114
  - mb\_sha512.h, 115
    - sha512\_flush\_job, 116
    - sha512\_flush\_job\_avx, 116
    - sha512\_flush\_job\_avx2, 117
    - sha512\_init\_mb\_mgr, 117
    - sha512\_init\_mb\_mgr\_x4, 117
    - sha512\_submit\_job, 118
    - sha512\_submit\_job\_avx, 118
    - sha512\_submit\_job\_avx2, 118
-



- md5\_ctx\_mgr\_flush
    - md5\_mb.h, [121](#)
  - md5\_ctx\_mgr\_flush\_avx
    - md5\_mb.h, [122](#)
  - md5\_ctx\_mgr\_flush\_avx2
    - md5\_mb.h, [122](#)
  - md5\_ctx\_mgr\_flush\_sse
    - md5\_mb.h, [122](#)
  - md5\_ctx\_mgr\_init
    - md5\_mb.h, [123](#)
  - md5\_ctx\_mgr\_init\_avx
    - md5\_mb.h, [123](#)
  - md5\_ctx\_mgr\_init\_avx2
    - md5\_mb.h, [123](#)
  - md5\_ctx\_mgr\_init\_sse
    - md5\_mb.h, [124](#)
  - md5\_ctx\_mgr\_submit
    - md5\_mb.h, [124](#)
  - md5\_ctx\_mgr\_submit\_avx
    - md5\_mb.h, [124](#)
  - md5\_ctx\_mgr\_submit\_avx2
    - md5\_mb.h, [125](#)
  - md5\_ctx\_mgr\_submit\_sse
    - md5\_mb.h, [125](#)
  - md5\_flush\_job
    - mb\_md5.h, [103](#)
  - md5\_flush\_job\_avx
    - mb\_md5.h, [103](#)
  - md5\_flush\_job\_avx2
    - mb\_md5.h, [103](#)
  - md5\_init\_mb\_mgr
    - mb\_md5.h, [104](#)
  - md5\_init\_mb\_mgr\_x8x2
    - mb\_md5.h, [104](#)
  - md5\_mb.h, [119](#)
    - md5\_ctx\_mgr\_flush, [121](#)
    - md5\_ctx\_mgr\_flush\_avx, [122](#)
    - md5\_ctx\_mgr\_flush\_avx2, [122](#)
    - md5\_ctx\_mgr\_flush\_sse, [122](#)
    - md5\_ctx\_mgr\_init, [123](#)
    - md5\_ctx\_mgr\_init\_avx, [123](#)
    - md5\_ctx\_mgr\_init\_avx2, [123](#)
    - md5\_ctx\_mgr\_init\_sse, [124](#)
    - md5\_ctx\_mgr\_submit, [124](#)
    - md5\_ctx\_mgr\_submit\_avx, [124](#)
    - md5\_ctx\_mgr\_submit\_avx2, [125](#)
    - md5\_ctx\_mgr\_submit\_sse, [125](#)
  - md5\_submit\_job
    - mb\_md5.h, [104](#)
  - md5\_submit\_job\_avx
    - mb\_md5.h, [105](#)
  - md5\_submit\_job\_avx2
    - mb\_md5.h, [105](#)
  - mem\_cmp\_avx
    - mem\_routines.h, [127](#)
  - mem\_cmp\_avx2
    - mem\_routines.h, [127](#)
  - mem\_cmp\_sse
    - mem\_routines.h, [127](#)
  - mem\_cpy\_avx
    - mem\_routines.h, [128](#)
  - mem\_cpy\_sse
    - mem\_routines.h, [128](#)
  - mem\_routines.h, [126](#)
    - mem\_cmp\_avx, [127](#)
    - mem\_cmp\_avx2, [127](#)
    - mem\_cmp\_sse, [127](#)
    - mem\_cpy\_avx, [128](#)
    - mem\_cpy\_sse, [128](#)
    - mem\_zero\_detect\_avx, [129](#)
  - mem\_zero\_detect\_avx
    - mem\_routines.h, [129](#)
  - memcpy\_inline.h, [129](#)
  - multi\_buffer.h
    - HASH\_CTX\_ERROR\_ALREADY\_COMPLETED, [130](#)
    - HASH\_CTX\_ERROR\_ALREADY\_PROCESSING, [130](#)
    - HASH\_CTX\_ERROR\_INVALID\_FLAGS, [130](#)
    - HASH\_CTX\_ERROR\_NONE, [130](#)
    - HASH\_CTX\_STS\_COMPLETE, [131](#)
    - HASH\_CTX\_STS\_IDLE, [131](#)
    - HASH\_CTX\_STS\_LAST, [131](#)
    - HASH\_CTX\_STS\_PROCESSING, [131](#)
    - HASH\_ENTIRE, [130](#)
    - HASH\_FIRST, [130](#)
    - HASH\_LAST, [130](#)
    - HASH\_UPDATE, [130](#)
    - STS\_BEING\_PROCESSED, [131](#)
    - STS\_COMPLETED, [131](#)
    - STS\_ERROR, [131](#)
    - STS\_INTERNAL\_ERROR, [131](#)
-

- STS\_UNKNOWN, 131
  - multi\_buffer.h, 129
    - HASH\_CTX\_ERROR, 130
    - HASH\_CTX\_FLAG, 130
    - HASH\_CTX\_STS, 130
    - JOB\_STS, 131
  - pq\_check
    - raid.h, 132
  - pq\_check\_base
    - raid.h, 133
  - pq\_check\_sse
    - raid.h, 133
  - pq\_gen
    - raid.h, 133
  - pq\_gen\_avx
    - raid.h, 134
  - pq\_gen\_avx2
    - raid.h, 134
  - pq\_gen\_base
    - raid.h, 135
  - pq\_gen\_sse
    - raid.h, 135
  - raid.h, 131
    - pq\_check, 132
    - pq\_check\_base, 133
    - pq\_check\_sse, 133
    - pq\_gen, 133
    - pq\_gen\_avx, 134
    - pq\_gen\_avx2, 134
    - pq\_gen\_base, 135
    - pq\_gen\_sse, 135
    - xor\_check, 135
    - xor\_check\_base, 136
    - xor\_check\_sse, 136
    - xor\_gen, 136
    - xor\_gen\_avx, 137
    - xor\_gen\_base, 137
    - xor\_gen\_sse, 138
  - STS\_BEING\_PROCESSED
    - multi\_buffer.h, 131
  - STS\_COMPLETED
    - multi\_buffer.h, 131
  - STS\_ERROR
    - multi\_buffer.h, 131
  - STS\_INTERNAL\_ERROR
    - multi\_buffer.h, 131
  - STS\_UNKNOWN
    - multi\_buffer.h, 131
  - SHA1\_ARGS\_X4, 38
  - SHA1\_ARGS\_X8, 38
  - SHA1\_HASH\_CTX, 39
  - SHA1\_HASH\_CTX\_MGR, 40
  - SHA1\_HMAC\_LANE\_DATA, 40
  - SHA1\_JOB, 41
  - SHA1\_LANE\_DATA, 41
  - SHA1\_MB\_ARGS\_X8, 42
  - SHA1\_MB\_JOB\_MGR, 42
  - SHA1\_MB\_MGR, 42
  - SHA1\_MB\_MGR\_X8, 43
  - SHA256\_ARGS\_X4, 44
  - SHA256\_ARGS\_X8, 44
  - SHA256\_HASH\_CTX, 45
  - SHA256\_HASH\_CTX\_MGR, 46
  - SHA256\_HMAC\_LANE\_DATA, 46
  - SHA256\_JOB, 47
  - SHA256\_LANE\_DATA, 47
  - SHA256\_MB\_ARGS\_X8, 48
  - SHA256\_MB\_JOB\_MGR, 48
  - SHA256\_MB\_MGR, 48
  - SHA256\_MB\_MGR\_X8, 49
  - SHA512\_ARGS\_X2, 50
  - SHA512\_ARGS\_X4, 50
  - SHA512\_HASH\_CTX, 51
  - SHA512\_HASH\_CTX\_MGR, 51
  - SHA512\_HMAC\_LANE\_DATA, 52
  - SHA512\_JOB, 52
  - SHA512\_LANE\_DATA, 53
  - SHA512\_MB\_ARGS\_X4, 53
  - SHA512\_MB\_JOB\_MGR, 54
  - SHA512\_MB\_MGR, 54
  - SHA512\_MB\_MGR\_X4, 55
  - sha.h, 138
    - sha1\_opt, 139
    - sha1\_update, 139
  - sha1\_ctx\_mgr\_flush
    - sha1\_mb.h, 142
  - sha1\_ctx\_mgr\_flush\_avx
    - sha1\_mb.h, 142
  - sha1\_ctx\_mgr\_flush\_avx2
    - sha1\_mb.h, 143
-

---

sha1\_ctx\_mgr\_flush\_sse  
  sha1\_mb.h, 143

sha1\_ctx\_mgr\_init  
  sha1\_mb.h, 143

sha1\_ctx\_mgr\_init\_avx  
  sha1\_mb.h, 144

sha1\_ctx\_mgr\_init\_avx2  
  sha1\_mb.h, 144

sha1\_ctx\_mgr\_init\_sse  
  sha1\_mb.h, 144

sha1\_ctx\_mgr\_submit  
  sha1\_mb.h, 145

sha1\_ctx\_mgr\_submit\_avx  
  sha1\_mb.h, 145

sha1\_ctx\_mgr\_submit\_avx2  
  sha1\_mb.h, 146

sha1\_ctx\_mgr\_submit\_sse  
  sha1\_mb.h, 146

sha1\_flush\_job  
  mb\_sha1.h, 107

sha1\_flush\_job\_avx  
  mb\_sha1.h, 107

sha1\_flush\_job\_avx2  
  mb\_sha1.h, 108

sha1\_init\_mb\_mgr  
  mb\_sha1.h, 108

sha1\_init\_mb\_mgr\_x8  
  mb\_sha1.h, 108

sha1\_mb.h, 139  
  sha1\_ctx\_mgr\_flush, 142  
  sha1\_ctx\_mgr\_flush\_avx, 142  
  sha1\_ctx\_mgr\_flush\_avx2, 143  
  sha1\_ctx\_mgr\_flush\_sse, 143  
  sha1\_ctx\_mgr\_init, 143  
  sha1\_ctx\_mgr\_init\_avx, 144  
  sha1\_ctx\_mgr\_init\_avx2, 144  
  sha1\_ctx\_mgr\_init\_sse, 144  
  sha1\_ctx\_mgr\_submit, 145  
  sha1\_ctx\_mgr\_submit\_avx, 145  
  sha1\_ctx\_mgr\_submit\_avx2, 146  
  sha1\_ctx\_mgr\_submit\_sse, 146

sha1\_opt  
  sha.h, 139

sha1\_submit\_job  
  mb\_sha1.h, 109

sha1\_submit\_job\_avx  
  mb\_sha1.h, 109

sha1\_submit\_job\_avx2  
  mb\_sha1.h, 110

sha1\_update  
  sha.h, 139

sha256\_ctx\_mgr\_flush  
  sha256\_mb.h, 149

sha256\_ctx\_mgr\_flush\_avx  
  sha256\_mb.h, 149

sha256\_ctx\_mgr\_flush\_avx2  
  sha256\_mb.h, 150

sha256\_ctx\_mgr\_flush\_sse  
  sha256\_mb.h, 150

sha256\_ctx\_mgr\_init  
  sha256\_mb.h, 151

sha256\_ctx\_mgr\_init\_avx  
  sha256\_mb.h, 151

sha256\_ctx\_mgr\_init\_avx2  
  sha256\_mb.h, 151

sha256\_ctx\_mgr\_init\_sse  
  sha256\_mb.h, 152

sha256\_ctx\_mgr\_submit  
  sha256\_mb.h, 152

sha256\_ctx\_mgr\_submit\_avx  
  sha256\_mb.h, 152

sha256\_ctx\_mgr\_submit\_avx2  
  sha256\_mb.h, 153

sha256\_ctx\_mgr\_submit\_sse  
  sha256\_mb.h, 153

sha256\_flush\_job  
  mb\_sha256.h, 112

sha256\_flush\_job\_avx  
  mb\_sha256.h, 112

sha256\_flush\_job\_avx2  
  mb\_sha256.h, 112

sha256\_init\_mb\_mgr  
  mb\_sha256.h, 113

sha256\_init\_mb\_mgr\_x8  
  mb\_sha256.h, 113

sha256\_mb.h, 147  
  sha256\_ctx\_mgr\_flush, 149  
  sha256\_ctx\_mgr\_flush\_avx, 149  
  sha256\_ctx\_mgr\_flush\_avx2, 150  
  sha256\_ctx\_mgr\_flush\_sse, 150  
  sha256\_ctx\_mgr\_init, 151  
  sha256\_ctx\_mgr\_init\_avx, 151

---

- sha256\_ctx\_mgr\_init\_avx2, 151
  - sha256\_ctx\_mgr\_init\_sse, 152
  - sha256\_ctx\_mgr\_submit, 152
  - sha256\_ctx\_mgr\_submit\_avx, 152
  - sha256\_ctx\_mgr\_submit\_avx2, 153
  - sha256\_ctx\_mgr\_submit\_sse, 153
  - sha256\_submit\_job
    - mb\_sha256.h, 113
  - sha256\_submit\_job\_avx
    - mb\_sha256.h, 114
  - sha256\_submit\_job\_avx2
    - mb\_sha256.h, 114
  - sha512\_ctx\_mgr\_flush
    - sha512\_mb.h, 157
  - sha512\_ctx\_mgr\_flush\_avx
    - sha512\_mb.h, 157
  - sha512\_ctx\_mgr\_flush\_avx2
    - sha512\_mb.h, 157
  - sha512\_ctx\_mgr\_flush\_sb\_sse4
    - sha512\_mb.h, 158
  - sha512\_ctx\_mgr\_flush\_sse
    - sha512\_mb.h, 158
  - sha512\_ctx\_mgr\_init
    - sha512\_mb.h, 159
  - sha512\_ctx\_mgr\_init\_avx
    - sha512\_mb.h, 159
  - sha512\_ctx\_mgr\_init\_avx2
    - sha512\_mb.h, 159
  - sha512\_ctx\_mgr\_init\_sb\_sse4
    - sha512\_mb.h, 160
  - sha512\_ctx\_mgr\_init\_sse
    - sha512\_mb.h, 160
  - sha512\_ctx\_mgr\_submit
    - sha512\_mb.h, 160
  - sha512\_ctx\_mgr\_submit\_avx
    - sha512\_mb.h, 161
  - sha512\_ctx\_mgr\_submit\_avx2
    - sha512\_mb.h, 161
  - sha512\_ctx\_mgr\_submit\_sb\_sse4
    - sha512\_mb.h, 162
  - sha512\_ctx\_mgr\_submit\_sse
    - sha512\_mb.h, 162
  - sha512\_flush\_job
    - mb\_sha512.h, 116
  - sha512\_flush\_job\_avx
    - mb\_sha512.h, 116
  - sha512\_flush\_job\_avx2
    - mb\_sha512.h, 117
  - sha512\_init\_mb\_mgr
    - mb\_sha512.h, 117
  - sha512\_init\_mb\_mgr\_x4
    - mb\_sha512.h, 117
  - sha512\_mb.h, 154
    - sha512\_ctx\_mgr\_flush, 157
    - sha512\_ctx\_mgr\_flush\_avx, 157
    - sha512\_ctx\_mgr\_flush\_avx2, 157
    - sha512\_ctx\_mgr\_flush\_sb\_sse4, 158
    - sha512\_ctx\_mgr\_flush\_sse, 158
    - sha512\_ctx\_mgr\_init, 159
    - sha512\_ctx\_mgr\_init\_avx, 159
    - sha512\_ctx\_mgr\_init\_avx2, 159
    - sha512\_ctx\_mgr\_init\_sb\_sse4, 160
    - sha512\_ctx\_mgr\_init\_sse, 160
    - sha512\_ctx\_mgr\_submit, 160
    - sha512\_ctx\_mgr\_submit\_avx, 161
    - sha512\_ctx\_mgr\_submit\_avx2, 161
    - sha512\_ctx\_mgr\_submit\_sb\_sse4, 162
    - sha512\_ctx\_mgr\_submit\_sse, 162
  - sha512\_submit\_job
    - mb\_sha512.h, 118
  - sha512\_submit\_job\_avx
    - mb\_sha512.h, 118
  - sha512\_submit\_job\_avx2
    - mb\_sha512.h, 118
  - types.h, 163
  - XTS\_AES\_128\_dec
    - aes\_xts.h, 58
  - XTS\_AES\_128\_dec\_expanded\_key
    - aes\_xts.h, 59
  - XTS\_AES\_128\_enc
    - aes\_xts.h, 59
  - XTS\_AES\_128\_enc\_expanded\_key
    - aes\_xts.h, 59
  - XTS\_AES\_256\_dec
    - aes\_xts.h, 60
  - XTS\_AES\_256\_dec\_expanded\_key
    - aes\_xts.h, 60
  - XTS\_AES\_256\_enc
    - aes\_xts.h, 61
  - XTS\_AES\_256\_enc\_expanded\_key
    - aes\_xts.h, 61
-

---

xor\_check  
    raid.h, [135](#)  
xor\_check\_base  
    raid.h, [136](#)  
xor\_check\_sse  
    raid.h, [136](#)  
xor\_gen  
    raid.h, [136](#)  
xor\_gen\_avx  
    raid.h, [137](#)  
xor\_gen\_base  
    raid.h, [137](#)  
xor\_gen\_sse  
    raid.h, [138](#)

---