



**EZ-PD™ CCGx Firmware Stack
API Reference Guide
Version 3.2.1**

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
<http://www.cypress.com>

Copyright © 2015-2018 Cypress Semiconductor Corporation. All rights reserved.

EZ-PD™ is a trademark of Cypress Semiconductor. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

The information in this document is subject to change without notice and should not be construed as a commitment by Cypress. While reasonable precautions have been taken, Cypress assumes no responsibility for any errors that may appear in this document. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Cypress. Made in the U.S.A.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

License Agreement

Please read the license agreement during SDK installation.

Contents

1 CCGx Firmware Stack: API Reference Guide	1
1.1 Introduction	1
1.1.1 USB Type-C Highlights	1
1.2 Cypress EZ-PD™ Type-C Controllers	1
1.2.1 CCGx Product Families	1
1.3 CCGx Firmware Stack	2
1.3.1 Firmware Stack Organization	2
2 CCGx Firmware Architecture	5
2.1 CCGx Firmware Solution Structure	6
2.2 Public API Summary	7
2.2.1 Device Policy Manager (DPM) API	7
2.2.2 Host Processor Interface (HPI) API	8
2.2.3 Application Layer API	9
2.2.3.1 Solution Layer Functions	10
2.2.4 Alternate Mode Manager API	10
2.2.5 Hardware Abstraction Layer (HAL) API	10
2.2.5.1 GPIO API	10
2.2.5.2 I2C Driver API	10
2.2.5.3 Flash Module API	11
2.2.5.4 Soft Timer API	11
2.2.6 Firmware Update API	11
3 Data Structure Index	13
3.1 Data Structures	13
4 File Index	17
4.1 File List	17
5 Data Structure Documentation	19
5.1 pd_do_t::ACT_CBL_VDO Struct Reference	19
5.1.1 Field Documentation	19
5.1.1.1 cbl_fw_ver	19

5.1.1.2	cbl_hw_ver	19
5.1.1.3	cbl_latency	19
5.1.1.4	cbl_term	20
5.1.1.5	max_vbus_VOLT	20
5.1.1.6	rsvd1	20
5.1.1.7	rsvd2	20
5.1.1.8	sop_dp	20
5.1.1.9	typec_abc	20
5.1.1.10	typec_plug	20
5.1.1.11	usb_ss_sup	20
5.1.1.12	vbus_cur	20
5.1.1.13	vbus_thru_cbl	21
5.1.1.14	vdo_version	21
5.2	alt_mode_evt_t::ALT_MODE_EVT Struct Reference	21
5.2.1	Detailed Description	21
5.2.2	Field Documentation	21
5.2.2.1	alt_mode	21
5.2.2.2	alt_mode_evt	21
5.2.2.3	data_role	21
5.2.2.4	svid	22
5.3	alt_mode_evt_t::ALT_MODE_EVT_DATA Struct Reference	22
5.3.1	Detailed Description	22
5.3.2	Field Documentation	22
5.3.2.1	evt_data	22
5.3.2.2	evt_type	22
5.4	alt_mode_evt_t Union Reference	22
5.4.1	Detailed Description	23
5.4.2	Field Documentation	23
5.4.2.1	alt_mode_event	23
5.4.2.2	alt_mode_event_data	23
5.4.2.3	val	23
5.5	alt_mode_hw_evt_t::ALT_MODE_HW_EVT Struct Reference	23
5.5.1	Detailed Description	23
5.5.2	Field Documentation	24
5.5.2.1	data_role	24
5.5.2.2	evt_data	24
5.5.2.3	hw_type	24
5.6	alt_mode_hw_evt_t Union Reference	24
5.6.1	Detailed Description	24
5.6.2	Field Documentation	24

5.6.2.1	hw_evt	24
5.6.2.2	val	25
5.7	alt_mode_info_t Struct Reference	25
5.7.1	Detailed Description	25
5.7.2	Field Documentation	25
5.7.2.1	alt_mode_id	25
5.7.2.2	app_evt_data	25
5.7.2.3	app_evt_needed	26
5.7.2.4	cbk	26
5.7.2.5	cbl_obj_pos	26
5.7.2.6	custom_att_obj_pos	26
5.7.2.7	eval_app_cmd	26
5.7.2.8	is_active	26
5.7.2.9	mode_state	26
5.7.2.10	obj_pos	26
5.7.2.11	set_mux_isolate	26
5.7.2.12	sop_state	27
5.7.2.13	uvdm_supp	27
5.7.2.14	vdm_header	27
5.7.2.15	vdo	27
5.7.2.16	vdo_max_numb	27
5.7.2.17	vdo_numb	27
5.8	alt_mode_reg_info_t Struct Reference	27
5.8.1	Detailed Description	28
5.8.2	Field Documentation	28
5.8.2.1	alt_mode_id	28
5.8.2.2	app_evt	28
5.8.2.3	atch_tgt_info	28
5.8.2.4	atch_type	28
5.8.2.5	cbl_sop_flag	28
5.8.2.6	data_role	28
5.8.2.7	svid_emca_vdo	28
5.8.2.8	svid_vdo	28
5.9	app_cbk_t Struct Reference	29
5.9.1	Detailed Description	29
5.9.2	Field Documentation	29
5.9.2.1	app_event_handler	30
5.9.2.2	eval_dr_swap	30
5.9.2.3	eval_fr_swap	30
5.9.2.4	eval_pr_swap	30

5.9.2.5 eval_rdo	30
5.9.2.6 eval_src_cap	30
5.9.2.7 eval_vconn_swap	30
5.9.2.8 eval_vdm	30
5.9.2.9 psnk_disable	30
5.9.2.10 psnk_enable	31
5.9.2.11 psnk_set_current	31
5.9.2.12 psnk_set_voltage	31
5.9.2.13 psrc_disable	31
5.9.2.14 psrc_enable	31
5.9.2.15 psrc_set_current	31
5.9.2.16 psrc_set_voltage	31
5.9.2.17 vbus_discharge_off	31
5.9.2.18 vbus_discharge_on	31
5.9.2.19 vbus_get_value	32
5.9.2.20 vbus_is_present	32
5.9.2.21 vconn_disable	32
5.9.2.22 vconn_enable	32
5.9.2.23 vconn_is_present	32
5.10 app_resp_t Struct Reference	32
5.10.1 Detailed Description	32
5.10.2 Field Documentation	32
5.10.2.1 req_status	33
5.10.2.2 resp_do	33
5.11 app_status_t Struct Reference	33
5.11.1 Detailed Description	33
5.11.2 Field Documentation	34
5.11.2.1 alt_mode_entered	34
5.11.2.2 alt_mode_trig_mask	34
5.11.2.3 app_resp	34
5.11.2.4 bc_12_src_disabled	34
5.11.2.5 cbl_disc_id_finished	34
5.11.2.6 cur_fb_enabled	34
5.11.2.7 fault_status	34
5.11.2.8 is_mux_busy	34
5.11.2.9 is_vbus_on	35
5.11.2.10 is_vconn_on	35
5.11.2.11 is_vdm_pending	35
5.11.2.12 ld_sw_ctrl	35
5.11.2.13 mux_poll_cbk	35

5.11.2.14 psnk_cur	35
5.11.2.15 psnk_volt	35
5.11.2.16 psrc_rising	35
5.11.2.17 psrc_volt	35
5.11.2.18 psrc_volt_old	36
5.11.2.19 pwr_ready_cbk	36
5.11.2.20 snk_dis_cbk	36
5.11.2.21 vdm_prcs_failed	36
5.11.2.22 vdm_resp	36
5.11.2.23 vdm_resp_cbk	36
5.11.2.24 vdm_retry_pending	36
5.11.2.25 vdm_task_en	36
5.11.2.26 vdm_version	36
5.12 atch_tgt_info_t Struct Reference	37
5.12.1 Detailed Description	37
5.12.2 Field Documentation	37
5.12.2.1 ama_vdo	37
5.12.2.2 cbl_svid	37
5.12.2.3 cbl_vdo	37
5.12.2.4 tgt_id_header	37
5.12.2.5 tgt_svid	37
5.13 bat_chg_params_t Struct Reference	38
5.13.1 Detailed Description	38
5.13.2 Field Documentation	38
5.13.2.1 reserved_0	38
5.13.2.2 reserved_1	38
5.13.2.3 table_len	38
5.13.2.4 vbatt_cutoff_volt	38
5.13.2.5 vbatt_dischg_en_volt	38
5.13.2.6 vbatt_max_cur	39
5.13.2.7 vbatt_max_volt	39
5.14 pd_do_t::BAT_SNK Struct Reference	39
5.14.1 Field Documentation	39
5.14.1.1 max_voltage	39
5.14.1.2 min_voltage	39
5.14.1.3 op_power	39
5.14.1.4 supply_type	39
5.15 pd_do_t::BAT_SRC Struct Reference	40
5.15.1 Field Documentation	40
5.15.1.1 max_power	40

5.15.1.2	max_voltage	40
5.15.1.3	min_voltage	40
5.15.1.4	supply_type	40
5.16	bb_handle_t Struct Reference	40
5.16.1	Detailed Description	41
5.16.2	Field Documentation	41
5.16.2.1	alt_status	41
5.16.2.2	bb_add_info	41
5.16.2.3	ep0_buffer	41
5.16.2.4	flashing_mode	41
5.16.2.5	num_alt_modes	41
5.16.2.6	queue_disable	41
5.16.2.7	queue_enable	42
5.16.2.8	queue_i2cm_enable	42
5.16.2.9	state	42
5.16.2.10	timeout	42
5.16.2.11	type	42
5.16.2.12	usb_configured	42
5.16.2.13	usb_i2cm_mode	42
5.16.2.14	usb_port	42
5.17	bb_settings_t Struct Reference	43
5.17.1	Detailed Description	43
5.17.2	Field Documentation	43
5.17.2.1	bb_always_on	43
5.17.2.2	bb_bos_dscr_offset	43
5.17.2.3	bb_bus_power	43
5.17.2.4	bb_ec_present	43
5.17.2.5	bb_enable	44
5.17.2.6	bb_option	44
5.17.2.7	bb_pid	44
5.17.2.8	bb_string_dscr_offset	44
5.17.2.9	bb_timeout	44
5.17.2.10	bb_unique_container_id	44
5.17.2.11	bb_vid	44
5.17.2.12	reserved_1	44
5.17.2.13	table_len	45
5.18	pd_do_t::BIST_DO Struct Reference	45
5.18.1	Field Documentation	45
5.18.1.1	mode	45
5.18.1.2	rsvd1	45

5.18.1.3	rsvd2	45
5.19	cc_state_t Union Reference	45
5.19.1	Detailed Description	46
5.19.2	Field Documentation	46
5.19.2.1	cc	46
5.19.2.2	state	46
5.20	ccg_timer_t Struct Reference	46
5.20.1	Detailed Description	46
5.20.2	Field Documentation	46
5.20.2.1	cb	46
5.20.2.2	count	47
5.20.2.3	id	47
5.20.2.4	period	47
5.21	chg_cfg_params_t Struct Reference	47
5.21.1	Detailed Description	47
5.21.2	Field Documentation	47
5.21.2.1	afc_src_cap_cnt	47
5.21.2.2	afc_src_caps	48
5.21.2.3	apple_src_id	48
5.21.2.4	qc_src_type	48
5.21.2.5	reserved_0	48
5.21.2.6	reserved_1	48
5.21.2.7	snk_sel	48
5.21.2.8	src_sel	48
5.21.2.9	table_len	48
5.22	comp_tbl_t Struct Reference	49
5.22.1	Detailed Description	49
5.22.2	Field Documentation	49
5.22.2.1	alt_mode_id	49
5.22.2.2	svid	49
5.23	contract_t Struct Reference	49
5.23.1	Detailed Description	49
5.23.2	Field Documentation	49
5.23.2.1	cur_pwr	50
5.23.2.2	max_volt	50
5.23.2.3	min_volt	50
5.24	pd_do_t::DP_CONFIG_VDO Struct Reference	50
5.24.1	Field Documentation	50
5.24.1.1	dfp_asgmt	50
5.24.1.2	rsvd1	50

5.24.1.3	rsvd2	50
5.24.1.4	sel_conf	51
5.24.1.5	sign	51
5.24.1.6	ufp_asgmt	51
5.25	pd_do_t::DP_STATUS_VDO Struct Reference	51
5.25.1	Field Documentation	51
5.25.1.1	dfp_ufp_conn	51
5.25.1.2	en	51
5.25.1.3	exit	52
5.25.1.4	hpd_irq	52
5.25.1.5	hpd_state	52
5.25.1.6	mult_fun	52
5.25.1.7	pwr_low	52
5.25.1.8	rsvd	52
5.25.1.9	usb_cfg	52
5.26	dpm_pd_cmd_buf_t Struct Reference	52
5.26.1	Detailed Description	53
5.26.2	Field Documentation	53
5.26.2.1	cmd_do	53
5.26.2.2	cmd_sop	53
5.26.2.3	dat_ptr	53
5.26.2.4	extd_hdr	53
5.26.2.5	extd_type	53
5.26.2.6	no_of_cmd_do	53
5.26.2.7	timeout	54
5.27	dpm_status_t Struct Reference	54
5.27.1	Detailed Description	56
5.27.2	Field Documentation	56
5.27.2.1	alert	56
5.27.2.2	app_cbk	56
5.27.2.3	attach	56
5.27.2.4	attached_dev	56
5.27.2.5	bist_cm2_enabled	57
5.27.2.6	bootup	57
5.27.2.7	cbl_dsc	57
5.27.2.8	cbl_mode_en	57
5.27.2.9	cbl_soft_reset_tried	57
5.27.2.10	cbl_state	57
5.27.2.11	cbl_type	57
5.27.2.12	cbl_vdm_version	57

5.27.2.13 cbl_vdo	57
5.27.2.14 cbl_wait	58
5.27.2.15 cc_live	58
5.27.2.16 cc_old_status	58
5.27.2.17 cc_rd_status	58
5.27.2.18 cc_status	58
5.27.2.19 cmd_p	58
5.27.2.20 connect	58
5.27.2.21 contract	58
5.27.2.22 contract_exist	58
5.27.2.23 cur_fb	59
5.27.2.24 cur_port_role	59
5.27.2.25 cur_port_type	59
5.27.2.26 cur_snk_max_min	59
5.27.2.27 cur_snk_pdo	59
5.27.2.28 cur_snk_pdo_count	59
5.27.2.29 cur_src_pdo	59
5.27.2.30 cur_src_pdo_count	59
5.27.2.31 db_support	59
5.27.2.32 dead_bat	60
5.27.2.33 dflt_port_role	60
5.27.2.34 dpm_cmd_buf	60
5.27.2.35 dpm_enabled	60
5.27.2.36 dpm_err_info	60
5.27.2.37 dpm_init	60
5.27.2.38 dpm_pd_cbk	60
5.27.2.39 dpm_pd_cmd	60
5.27.2.40 dpm_pd_cmd_active	60
5.27.2.41 dpm_safe_disable	61
5.27.2.42 dpm_typec_cbk	61
5.27.2.43 dpm_typec_cmd	61
5.27.2.44 dpm_typec_cmd_active	61
5.27.2.45 drp_period	61
5.27.2.46 emca_present	61
5.27.2.47 err_recov	61
5.27.2.48 ext_src_cap	61
5.27.2.49 fault_active	61
5.27.2.50 fr_rx_disabled	62
5.27.2.51 fr_rx_en_live	62
5.27.2.52 fr_tx_disabled	62

5.27.2.53 fr_tx_en_live	62
5.27.2.54 frs_enable	62
5.27.2.55 is_snk_bat	62
5.27.2.56 is_src_bat	62
5.27.2.57 non_intr_response	62
5.27.2.58 padval	62
5.27.2.59 pd_connected	63
5.27.2.60 pd_disabled	63
5.27.2.61 pd_support	63
5.27.2.62 pe_evt	63
5.27.2.63 pe_fsm_state	63
5.27.2.64 polarity	63
5.27.2.65 port_disable	63
5.27.2.66 port_role	63
5.27.2.67 port_status	63
5.27.2.68 pps_status	64
5.27.2.69 ra_present	64
5.27.2.70 reserved_3	64
5.27.2.71 rev_pol	64
5.27.2.72 role_at_connect	64
5.27.2.73 rp_supported	64
5.27.2.74 skip_scan	64
5.27.2.75 snk_cur_level	64
5.27.2.76 snk_max_min	65
5.27.2.77 snk_pdo	65
5.27.2.78 snk_pdo_count	65
5.27.2.79 snk_pdo_mask	65
5.27.2.80 snk_period	65
5.27.2.81 snk_rdo	65
5.27.2.82 snk_rp_detach_en	65
5.27.2.83 snk_sel_pdo	65
5.27.2.84 snk_usb_comm_en	65
5.27.2.85 snk_usb_susp_en	66
5.27.2.86 spec_rev_cbl	66
5.27.2.87 spec_rev_peer	66
5.27.2.88 spec_rev_sop_live	66
5.27.2.89 spec_rev_sop_prime_live	66
5.27.2.90 src_cap_p	66
5.27.2.91 src_cur_level	66
5.27.2.92 src_cur_level_live	66

5.27.2.93 src_cur_rdo	66
5.27.2.94 src_last_rdo	67
5.27.2.95 src_pdo	67
5.27.2.96 src_pdo_count	67
5.27.2.97 src_pdo_mask	67
5.27.2.98 src_period	67
5.27.2.99 src_rdo	67
5.27.2.100src_sel_pdo	67
5.27.2.101swap_response	67
5.27.2.102toggle	68
5.27.2.103try_src_snk	68
5.27.2.104typec_evt	68
5.27.2.105typec_fsm_state	68
5.27.2.106unchunk_sup_live	68
5.27.2.107unchunk_sup_peer	68
5.27.2.108vconn_logical	68
5.27.2.109vconn_retain	68
5.28 pd_do_t::FIXED_SNK Struct Reference	68
5.28.1 Field Documentation	69
5.28.1.1 dr_swap	69
5.28.1.2 dual_role_power	69
5.28.1.3 extPowered	69
5.28.1.4 fr_swap	69
5.28.1.5 high_cap	69
5.28.1.6 op_current	69
5.28.1.7 rsrvd	70
5.28.1.8 supply_type	70
5.28.1.9 usb_comm_cap	70
5.28.1.10 voltage	70
5.29 pd_do_t::FIXED_SRC Struct Reference	70
5.29.1 Field Documentation	70
5.29.1.1 dr_swap	70
5.29.1.2 dual_role_power	71
5.29.1.3 extPowered	71
5.29.1.4 max_current	71
5.29.1.5 pk_current	71
5.29.1.6 reserved	71
5.29.1.7 supply_type	71
5.29.1.8 unchunk_sup	71
5.29.1.9 usb_comm_cap	71

5.29.1.10 <code>usb_suspend_sup</code>	71
5.29.1.11 <code>voltage</code>	72
5.30 <code>fw_img_status_t</code> Union Reference	72
5.30.1 Detailed Description	72
5.30.2 Field Documentation	72
5.30.2.1 <code>status</code>	72
5.30.2.2 <code>val</code>	72
5.31 <code>fw_img_status_t::fw_mode_reason_t</code> Struct Reference	73
5.31.1 Field Documentation	73
5.31.1.1 <code>boot_mode_request</code>	73
5.31.1.2 <code>fw1_invalid</code>	73
5.31.1.3 <code>fw2_invalid</code>	73
5.31.1.4 <code>reserved</code>	73
5.31.1.5 <code>reserved1</code>	73
5.32 <code>i2c_scb_config_t</code> Struct Reference	73
5.32.1 Detailed Description	74
5.32.2 Field Documentation	74
5.32.2.1 <code>buf_size</code>	74
5.32.2.2 <code>buffer</code>	74
5.32.2.3 <code>cb_fun_ptr</code>	74
5.32.2.4 <code>clock_freq</code>	74
5.32.2.5 <code>i2c_state</code>	74
5.32.2.6 <code>i2c_write_count</code>	75
5.32.2.7 <code>mode</code>	75
5.32.2.8 <code>slave_address</code>	75
5.32.2.9 <code>slave_mask</code>	75
5.33 <code>ocp_settings_t</code> Struct Reference	75
5.33.1 Detailed Description	75
5.33.2 Field Documentation	75
5.33.2.1 <code>cs_res</code>	76
5.33.2.2 <code>debounce</code>	76
5.33.2.3 <code>debounce2</code>	76
5.33.2.4 <code>retry_cnt</code>	76
5.33.2.5 <code>sense_res</code>	76
5.33.2.6 <code>table_len</code>	76
5.33.2.7 <code>threshold</code>	76
5.33.2.8 <code>threshold2</code>	76
5.34 <code>otp_settings_t</code> Struct Reference	76
5.34.1 Detailed Description	77
5.34.2 Field Documentation	77

5.34.2.1	cutoff_volt	77
5.34.2.2	debounce	77
5.34.2.3	reserved_0	77
5.34.2.4	restart_volt	77
5.34.2.5	table_len	77
5.34.2.6	therm_type	77
5.35	ovp_settings_t Struct Reference	78
5.35.1	Detailed Description	78
5.35.2	Field Documentation	78
5.35.2.1	debounce	78
5.35.2.2	retry_cnt	78
5.35.2.3	table_len	78
5.35.2.4	threshold	78
5.36	pd_do_t::PAS_CBL_VDO Struct Reference	79
5.36.1	Field Documentation	79
5.36.1.1	cbl_fw_ver	79
5.36.1.2	cbl_hw_ver	79
5.36.1.3	cbl_latency	79
5.36.1.4	cbl_term	79
5.36.1.5	max_vbus_volt	79
5.36.1.6	rsvd1	80
5.36.1.7	rsvd2	80
5.36.1.8	rsvd3	80
5.36.1.9	typec_abc	80
5.36.1.10	typec_plug	80
5.36.1.11	usb_ss_sup	80
5.36.1.12	vbus_cur	80
5.36.1.13	vdo_version	80
5.37	pd_config_t Struct Reference	80
5.37.1	Detailed Description	81
5.37.2	Field Documentation	81
5.37.2.1	application	81
5.37.2.2	flash_checksum	81
5.37.2.3	flashing_mode	81
5.37.2.4	flashing_vid	81
5.37.2.5	mfg_info_length	82
5.37.2.6	mfg_info_offset	82
5.37.2.7	pd_port_cnt	82
5.37.2.8	port_conf	82
5.37.2.9	reserved_0	82

5.37.2.10 reserved_1	82
5.37.2.11 reserved_2	82
5.37.2.12 table_checksum	82
5.37.2.13 table_sign	82
5.37.2.14 table_size	83
5.37.2.15 table_type	83
5.37.2.16 table_version	83
5.38 pd_contract_info_t Struct Reference	83
5.38.1 Detailed Description	83
5.38.2 Field Documentation	83
5.38.2.1 rdo	83
5.38.2.2 status	83
5.39 pd_do_t Union Reference	84
5.39.1 Detailed Description	85
5.39.2 Field Documentation	85
5.39.2.1 act_cbl_vdo	85
5.39.2.2 bat_snk	85
5.39.2.3 bat_src	85
5.39.2.4 bist_do	85
5.39.2.5 dp_cfg_vdo	86
5.39.2.6 dp_stat_vdo	86
5.39.2.7 fixed_snk	86
5.39.2.8 fixed_src	86
5.39.2.9 pas_cbl_vdo	86
5.39.2.10 qc_4_0_data_vdo	86
5.39.2.11 rdo_bat	86
5.39.2.12 rdo_bat_gvb	86
5.39.2.13 rdo_fix_var	86
5.39.2.14 rdo_fix_var_gvb	87
5.39.2.15 rdo_gen	87
5.39.2.16 rdo_gen_gvb	87
5.39.2.17 src_gen	87
5.39.2.18 std_ama_vdo	87
5.39.2.19 std_ama_vdo_pd3	87
5.39.2.20 std_cbl_vdo	87
5.39.2.21 std_cert_vdo	87
5.39.2.22 std_dp_vdo	87
5.39.2.23 std_id_hdr	88
5.39.2.24 std_prod_vdo	88
5.39.2.25 std_svid_res	88

5.39.2.26 std_vdm_hdr	88
5.39.2.27 ustd_qc_4_0_hdr	88
5.39.2.28 ustd_vdm_hdr	88
5.39.2.29 val	88
5.39.2.30 var_snk	88
5.39.2.31 var_src	88
5.40 pd_extd_hdr_t Union Reference	89
5.40.1 Detailed Description	89
5.40.2 Field Documentation	89
5.40.2.1 chunk_no	89
5.40.2.2 chunked	89
5.40.2.3 data_size	89
5.40.2.4 extd	89
5.40.2.5 request	89
5.40.2.6 rsvd1	90
5.40.2.7 val	90
5.41 pd_hdr_t::PD_HDR Struct Reference	90
5.41.1 Field Documentation	90
5.41.1.1 chunk_no	90
5.41.1.2 chunked	90
5.41.1.3 data_role	90
5.41.1.4 data_size	91
5.41.1.5 extd	91
5.41.1.6 len	91
5.41.1.7 msg_id	91
5.41.1.8 msg_type	91
5.41.1.9 pwr_role	91
5.41.1.10 request	91
5.41.1.11 rsvd1	91
5.41.1.12 spec_rev	91
5.42 pd_hdr_t Union Reference	92
5.42.1 Detailed Description	92
5.42.2 Field Documentation	92
5.42.2.1 hdr	92
5.42.2.2 val	92
5.43 pd_packet_extd_t Struct Reference	92
5.43.1 Detailed Description	93
5.43.2 Field Documentation	93
5.43.2.1 dat	93
5.43.2.2 data_role	93

5.43.2.3	hdr	93
5.43.2.4	len	93
5.43.2.5	msg	93
5.43.2.6	sop	93
5.44	pd_packet_t Struct Reference	93
5.44.1	Detailed Description	94
5.44.2	Field Documentation	94
5.44.2.1	dat	94
5.44.2.2	data_role	94
5.44.2.3	hdr	94
5.44.2.4	len	94
5.44.2.5	msg	94
5.44.2.6	sop	94
5.45	pd_port_config_t Struct Reference	95
5.45.1	Detailed Description	96
5.45.2	Field Documentation	96
5.45.2.1	bat_chg_tbl_offset	96
5.45.2.2	bb_tbl_offset	96
5.45.2.3	cable_disc_en	96
5.45.2.4	chg_cfg_tbl_offset	96
5.45.2.5	current_level	96
5.45.2.6	dead_bat_support	97
5.45.2.7	default_role	97
5.45.2.8	default_sink_pdo_mask	97
5.45.2.9	default_src_pdo_mask	97
5.45.2.10	dock_cfg_tbl_offset	97
5.45.2.11	dp_config_supported	97
5.45.2.12	dp_mode_trigger	97
5.45.2.13	dp_mux_control	97
5.45.2.14	dp_oper	97
5.45.2.15	dp_pref_mode	98
5.45.2.16	drp_toggle_en	98
5.45.2.17	err_recovery_en	98
5.45.2.18	ext_src_cap_length	98
5.45.2.19	ext_src_cap_offset	98
5.45.2.20	frs_enable	98
5.45.2.21	id_vdm_length	98
5.45.2.22	id_vdm_offset	98
5.45.2.23	is_snk_bat	98
5.45.2.24	is_src_bat	99

5.45.2.25 mode_vdm_length	99
5.45.2.26 mode_vdm_offset	99
5.45.2.27 ocp_tbl_offset	99
5.45.2.28 otp_tbl_offset	99
5.45.2.29 ovp_tbl_offset	99
5.45.2.30 pd_operation_en	99
5.45.2.31 port_disable	99
5.45.2.32 port_role	99
5.45.2.33 protection_enable	100
5.45.2.34 pwr_tbl_offset	100
5.45.2.35 ra_timeout	100
5.45.2.36 reserved_0	100
5.45.2.37 reserved_1	100
5.45.2.38 reserved_10	100
5.45.2.39 reserved_11	100
5.45.2.40 reserved_3	100
5.45.2.41 reserved_4	100
5.45.2.42 reserved_6	101
5.45.2.43 reserved_8	101
5.45.2.44 reserved_9	101
5.45.2.45 rp_supported	101
5.45.2.46 scp_tbl_offset	101
5.45.2.47 snk_pdo_cnt	101
5.45.2.48 snk_pdo_list	101
5.45.2.49 snk_pdo_max_min_current_pwr	101
5.45.2.50 snk_usb_comm_en	101
5.45.2.51 snk_usb_susp_en	102
5.45.2.52 spm_cfg_tbl_offset	102
5.45.2.53 src_pdo_cnt	102
5.45.2.54 src_pdo_list	102
5.45.2.55 svid_vdm_length	102
5.45.2.56 svid_vdm_offset	102
5.45.2.57 swap_response	102
5.45.2.58 try_src_en	102
5.45.2.59 type_a_chg_tbl_offset	102
5.45.2.60 type_a_enable	103
5.45.2.61 type_a_pwr_tbl_offset	103
5.45.2.62 uvp_tbl_offset	103
5.45.2.63 vconn_ocp_tbl_offset	103
5.45.2.64 vconn_retain	103

5.46 pd_port_status_t::PD_PORT_STAT Struct Reference	103
5.46.1 Field Documentation	104
5.46.1.1 ccg_spec_rev	104
5.46.1.2 contract_exist	104
5.46.1.3 cur_data_role	104
5.46.1.4 cur_power_role	104
5.46.1.5 dflt_data_pref	104
5.46.1.6 dflt_data_role	104
5.46.1.7 dflt_power_pref	104
5.46.1.8 dflt_power_role	104
5.46.1.9 emca_present	105
5.46.1.10 emca_spec_rev	105
5.46.1.11 min_state	105
5.46.1.12 pe_rdy	105
5.46.1.13 peer_pd3_supp	105
5.46.1.14 peer_unchunk_supp	105
5.46.1.15 reserved0	105
5.46.1.16 reserved2	105
5.46.1.17 rp_status	105
5.46.1.18 vconn_on	106
5.46.1.19 vconn_src	106
5.47 pd_port_status_t Union Reference	106
5.47.1 Detailed Description	106
5.47.2 Field Documentation	106
5.47.2.1 status	106
5.47.2.2 val	106
5.48 pd_power_status_t Struct Reference	107
5.48.1 Detailed Description	107
5.48.2 Field Documentation	107
5.48.2.1 battery_input	107
5.48.2.2 dummy	107
5.48.2.3 event_flags	107
5.48.2.4 intl_temperature	107
5.48.2.5 present_input	107
5.48.2.6 temp_status	108
5.49 pwr_params_t Struct Reference	108
5.49.1 Detailed Description	108
5.49.2 Field Documentation	108
5.49.2.1 fb_ctrl_r1	108
5.49.2.2 fb_ctrl_r2	108

5.49.2.3	<code>fb_type</code>	108
5.49.2.4	<code>table_len</code>	109
5.49.2.5	<code>vbus_dfilt_volt</code>	109
5.49.2.6	<code>vbus_max_volt</code>	109
5.49.2.7	<code>vbus_min_volt</code>	109
5.50	<code>pd_do_t::QC_4_0_DATA_VDO</code> Struct Reference	109
5.50.1	Field Documentation	109
5.50.1.1	<code>data_0</code>	109
5.50.1.2	<code>data_1</code>	109
5.50.1.3	<code>data_2</code>	110
5.50.1.4	<code>data_3</code>	110
5.51	<code>pd_do_t::RDO_BAT</code> Struct Reference	110
5.51.1	Field Documentation	110
5.51.1.1	<code>cap_mismatch</code>	110
5.51.1.2	<code>give_back_flag</code>	110
5.51.1.3	<code>max_op_power</code>	110
5.51.1.4	<code>no_usb_suspend</code>	111
5.51.1.5	<code>obj_pos</code>	111
5.51.1.6	<code>op_power</code>	111
5.51.1.7	<code>rsrvd1</code>	111
5.51.1.8	<code>rsrvd2</code>	111
5.51.1.9	<code>unchunk_sup</code>	111
5.51.1.10	<code>usb_comm_cap</code>	111
5.52	<code>pd_do_t::RDO_BAT_GIVEBACK</code> Struct Reference	111
5.52.1	Field Documentation	112
5.52.1.1	<code>cap_mismatch</code>	112
5.52.1.2	<code>give_back_flag</code>	112
5.52.1.3	<code>min_op_power</code>	112
5.52.1.4	<code>no_usb_suspend</code>	112
5.52.1.5	<code>obj_pos</code>	112
5.52.1.6	<code>op_power</code>	112
5.52.1.7	<code>rsrvd1</code>	112
5.52.1.8	<code>rsrvd2</code>	113
5.52.1.9	<code>unchunk_sup</code>	113
5.52.1.10	<code>usb_comm_cap</code>	113
5.53	<code>pd_do_t::RDO_FIXED_VAR</code> Struct Reference	113
5.53.1	Field Documentation	113
5.53.1.1	<code>cap_mismatch</code>	113
5.53.1.2	<code>give_back_flag</code>	113
5.53.1.3	<code>max_op_current</code>	114

5.53.1.4 no_usb_suspend	114
5.53.1.5 obj_pos	114
5.53.1.6 op_current	114
5.53.1.7 rsrvd1	114
5.53.1.8 rsrvd2	114
5.53.1.9 unchunk_sup	114
5.53.1.10 usb_comm_cap	114
5.54 pd_do_t::RDO_FIXED_VAR_GIVEBACK Struct Reference	114
5.54.1 Field Documentation	115
5.54.1.1 cap_mismatch	115
5.54.1.2 give_back_flag	115
5.54.1.3 min_op_current	115
5.54.1.4 no_usb_suspend	115
5.54.1.5 obj_pos	115
5.54.1.6 op_current	115
5.54.1.7 rsrvd1	116
5.54.1.8 rsrvd2	116
5.54.1.9 unchunk_sup	116
5.54.1.10 usb_comm_cap	116
5.55 pd_do_t::RDO_GEN Struct Reference	116
5.55.1 Field Documentation	116
5.55.1.1 cap_mismatch	116
5.55.1.2 give_back_flag	117
5.55.1.3 min_max_power_cur	117
5.55.1.4 no_usb_suspend	117
5.55.1.5 obj_pos	117
5.55.1.6 op_power_cur	117
5.55.1.7 rsrvd1	117
5.55.1.8 rsrvd2	117
5.55.1.9 unchunk_sup	117
5.55.1.10 usb_comm_cap	117
5.56 pd_do_t::RDO_GEN_GVB Struct Reference	118
5.56.1 Field Documentation	118
5.56.1.1 cap_mismatch	118
5.56.1.2 give_back_flag	118
5.56.1.3 max_power_cur	118
5.56.1.4 no_usb_suspend	118
5.56.1.5 obj_pos	118
5.56.1.6 op_power_cur	118
5.56.1.7 rsrvd1	119

5.56.1.8	rsrvd2	119
5.56.1.9	unchunk_sup	119
5.56.1.10	usb_comm_cap	119
5.57	ridge_reg_t Union Reference	119
5.57.1	Detailed Description	119
5.57.2	Field Documentation	119
5.57.2.1	ridge_stat	120
5.57.2.2	usbpd_cmd_reg	120
5.57.2.3	val	120
5.58	scp_settings_t Struct Reference	120
5.58.1	Detailed Description	120
5.58.2	Field Documentation	120
5.58.2.1	debounce	120
5.58.2.2	retry_cnt	120
5.58.2.3	table_len	121
5.58.2.4	threshold	121
5.59	pd_do_t::SRC_GEN Struct Reference	121
5.59.1	Field Documentation	121
5.59.1.1	max_cur_power	121
5.59.1.2	max_voltage	121
5.59.1.3	min_voltage	121
5.59.1.4	supply_type	121
5.60	pd_do_t::STD_AMA_VDO Struct Reference	122
5.60.1	Field Documentation	122
5.60.1.1	ama_fw_ver	122
5.60.1.2	ama_hw_ver	122
5.60.1.3	rsvd1	122
5.60.1.4	ssrx1	122
5.60.1.5	ssrx2	122
5.60.1.6	sstx1	122
5.60.1.7	sstx2	123
5.60.1.8	usb_ss_sup	123
5.60.1.9	vbus_req	123
5.60.1.10	vcon_pwr	123
5.60.1.11	vcon_req	123
5.61	pd_do_t::STD_AMA_VDO_PD3 Struct Reference	123
5.61.1	Field Documentation	123
5.61.1.1	ama_fw_ver	123
5.61.1.2	ama_hw_ver	124
5.61.1.3	rsvd1	124

5.61.1.4	usb_ss_sup	124
5.61.1.5	vbus_req	124
5.61.1.6	vcon_pwr	124
5.61.1.7	vcon_req	124
5.61.1.8	vdo_version	124
5.62	pd_do_t::STD_CBL_VDO Struct Reference	124
5.62.1	Field Documentation	125
5.62.1.1	cbl_fw_ver	125
5.62.1.2	cbl_hw_ver	125
5.62.1.3	cbl_latency	125
5.62.1.4	cbl_term	125
5.62.1.5	rsvd1	125
5.62.1.6	sop_dp	125
5.62.1.7	ssrx1	125
5.62.1.8	ssrx2	126
5.62.1.9	sstx1	126
5.62.1.10	sstx2	126
5.62.1.11	typec_abc	126
5.62.1.12	typec_plug	126
5.62.1.13	usb_ss_sup	126
5.62.1.14	vbus_cur	126
5.62.1.15	vbus_thru_cbl	126
5.63	pd_do_t::STD_CERT_VDO Struct Reference	126
5.63.1	Field Documentation	127
5.63.1.1	usb_xid	127
5.64	pd_do_t::STD_DP_VDO Struct Reference	127
5.64.1	Field Documentation	127
5.64.1.1	dfp_d_pin	127
5.64.1.2	port_cap	127
5.64.1.3	recep	127
5.64.1.4	rsvd	128
5.64.1.5	signal	128
5.64.1.6	ufp_d_pin	128
5.64.1.7	usb2_0	128
5.65	pd_do_t::STD_PROD_VDO Struct Reference	128
5.65.1	Field Documentation	128
5.65.1.1	bcd_dev	128
5.65.1.2	usb_pid	128
5.66	pd_do_t::STD_SVID_RESP_VDO Struct Reference	129
5.66.1	Field Documentation	129

5.66.1.1	svid_n	129
5.66.1.2	svid_n1	129
5.67	pd_do_t::STD_VDM_HDR Struct Reference	129
5.67.1	Field Documentation	129
5.67.1.1	cmd	129
5.67.1.2	cmd_type	130
5.67.1.3	obj_pos	130
5.67.1.4	rsvd1	130
5.67.1.5	rsvd2	130
5.67.1.6	st_ver	130
5.67.1.7	svid	130
5.67.1.8	vdm_type	130
5.68	pd_do_t::STD_VDM_ID_HDR Struct Reference	130
5.68.1	Field Documentation	131
5.68.1.1	mod_support	131
5.68.1.2	prod_type	131
5.68.1.3	usb_dev	131
5.68.1.4	usb_host	131
5.68.1.5	usb_vid	131
5.69	sys_fw_metadata_t Struct Reference	131
5.69.1	Detailed Description	132
5.69.2	Field Documentation	132
5.69.2.1	active_boot_app	132
5.69.2.2	boot_app_id	132
5.69.2.3	boot_app_ver_status	132
5.69.2.4	boot_app_version	132
5.69.2.5	boot_last_row	132
5.69.2.6	boot_seq	132
5.69.2.7	fw_checksum	133
5.69.2.8	fw_entry	133
5.69.2.9	fw_size	133
5.69.2.10	fw_version	133
5.69.2.11	metadata_valid	133
5.69.2.12	reserved1	133
5.69.2.13	reserved2	133
5.70	usb_config_t Struct Reference	133
5.70.1	Detailed Description	134
5.70.2	Field Documentation	134
5.70.2.1	bus_power	134
5.70.2.2	class_rqt_cb	134

5.70.2.3	event_cb	134
5.70.2.4	fallback_rqt_cb	134
5.70.2.5	get_dscr_cb	134
5.70.2.6	vendor_rqt_cb	134
5.71	usb_ep_handle_t Struct Reference	135
5.71.1	Detailed Description	135
5.71.2	Field Documentation	135
5.71.2.1	enabled	135
5.71.2.2	is_out	135
5.71.2.3	toggle	135
5.72	usb_handle_t Struct Reference	135
5.72.1	Detailed Description	136
5.72.2	Field Documentation	136
5.72.2.1	active_alt_inf	136
5.72.2.2	active_cfg	136
5.72.2.3	cfg	136
5.72.2.4	dev_stat	136
5.72.2.5	ep0_buffer	136
5.72.2.6	ep0_last	137
5.72.2.7	ep0_length	137
5.72.2.8	ep0_state	137
5.72.2.9	ep0_toggle	137
5.72.2.10	ep0_xfer_cb	137
5.72.2.11	ep0_zlp_rqd	137
5.72.2.12	ep_handle	137
5.72.2.13	int_event	137
5.72.2.14	prev_state	137
5.72.2.15	setup_pkt	138
5.72.2.16	sof_data	138
5.72.2.17	state	138
5.72.2.18	suspend_check	138
5.73	usb_i2cm_t Struct Reference	138
5.73.1	Detailed Description	138
5.73.2	Field Documentation	138
5.73.2.1	count	139
5.73.2.2	ctrl	139
5.73.2.3	length	139
5.73.2.4	preamble	139
5.73.2.5	preamble_state	139
5.73.2.6	state	139

5.73.2.7	status	139
5.74	usb_setup_pkt_t Struct Reference	139
5.74.1	Detailed Description	140
5.74.2	Field Documentation	140
5.74.2.1	attrib	140
5.74.2.2	cmd	140
5.74.2.3	index	140
5.74.2.4	length	140
5.74.2.5	value	140
5.75	USBARB16_REGS_T Struct Reference	140
5.75.1	Detailed Description	141
5.76	USBARB_REGS_T Struct Reference	141
5.76.1	Detailed Description	141
5.77	ridge_reg_t::USBPD_CMD_REG Struct Reference	141
5.77.1	Detailed Description	142
5.77.2	Field Documentation	142
5.77.2.1	dp_host_conn	142
5.77.2.2	hpd_irq	142
5.77.2.3	hpd_lvl	142
5.77.2.4	i2c_int_ack	142
5.77.2.5	irq_ack	142
5.77.2.6	rsvd2	143
5.77.2.7	rsvd3	143
5.77.2.8	rsvd4	143
5.77.2.9	soft_rst	143
5.77.2.10	tbt_host_conn	143
5.77.2.11	usb_host_conn	143
5.78	ridge_reg_t::USBPD_STATUS_REG Struct Reference	143
5.78.1	Detailed Description	144
5.78.2	Field Documentation	144
5.78.2.1	act_link_train	144
5.78.2.2	active_cbl	144
5.78.2.3	cbl_type	144
5.78.2.4	conn_orien	144
5.78.2.5	data_conn_pres	145
5.78.2.6	dbg_acc_mode	145
5.78.2.7	dbg_alt_m_conn	145
5.78.2.8	dp_conn	145
5.78.2.9	dp_pin_assign	145
5.78.2.10	dp_role	145

5.78.2.11 force_lsx	145
5.78.2.12 hpd_irq	145
5.78.2.13 hpd_lvl	145
5.78.2.14 interrupt_ack	146
5.78.2.15 irq_ack	146
5.78.2.16 ovc_indn	146
5.78.2.17 pro_dock_detect	146
5.78.2.18 pwr	146
5.78.2.19 rsvd	146
5.78.2.20 rsvd4	146
5.78.2.21 tbt_cbl_gen	146
5.78.2.22 tbt_cbl_spd	146
5.78.2.23 tbt_conn	147
5.78.2.24 tbt_type	147
5.78.2.25 usb2_conn	147
5.78.2.26 usb3_conn	147
5.78.2.27 usb3_speed	147
5.78.2.28 usb_dr	147
5.79 USBSIE_REGS_T Struct Reference	147
5.79.1 Detailed Description	148
5.80 pd_do_t::USTD_QC_4_0_HDR Struct Reference	148
5.80.1 Field Documentation	148
5.80.1.1 cmd_0	148
5.80.1.2 cmd_1	148
5.80.1.3 svid	148
5.80.1.4 vdm_type	148
5.81 pd_do_t::USTD_VDM_HDR Struct Reference	148
5.81.1 Field Documentation	149
5.81.1.1 cmd	149
5.81.1.2 cmd_type	149
5.81.1.3 rsvd1	149
5.81.1.4 seq_num	149
5.81.1.5 svid	149
5.81.1.6 vdm_type	149
5.81.1.7 vdm_ver	149
5.82 uvp_settings_t Struct Reference	150
5.82.1 Detailed Description	150
5.82.2 Field Documentation	150
5.82.2.1 debounce	150
5.82.2.2 retry_cnt	150

5.82.2.3	table_len	150
5.82.2.4	threshold	150
5.83	pd_do_t::VAR_SNK Struct Reference	150
5.83.1	Field Documentation	151
5.83.1.1	max_voltage	151
5.83.1.2	min_voltage	151
5.83.1.3	op_current	151
5.83.1.4	supply_type	151
5.84	pd_do_t::VAR_SRC Struct Reference	151
5.84.1	Field Documentation	151
5.84.1.1	max_current	151
5.84.1.2	max_voltage	152
5.84.1.3	min_voltage	152
5.84.1.4	supply_type	152
5.85	vconn_ocp_settings_t Struct Reference	152
5.85.1	Detailed Description	152
5.85.2	Field Documentation	152
5.85.2.1	debounce	152
5.85.2.2	reserved_0	152
5.85.2.3	table_len	153
5.85.2.4	threshold	153
5.86	vdm_msg_info_t Struct Reference	153
5.86.1	Detailed Description	153
5.86.2	Field Documentation	153
5.86.2.1	sop_type	153
5.86.2.2	vdm_header	153
5.86.2.3	vdo	153
5.86.2.4	vdo_numb	154
5.87	vdm_resp_t Struct Reference	154
5.87.1	Detailed Description	154
5.87.2	Field Documentation	154
5.87.2.1	do_count	154
5.87.2.2	no_resp	154
5.87.2.3	resp_buf	154
6	File Documentation	155
6.1	app/alt_mode/alt_mode_hw.h File Reference	155
6.1.1	Detailed Description	156
6.1.2	Enumeration Type Documentation	156
6.1.2.1	alt_mode_hw_t	156

6.1.2.2	<code>mux_poll_status_t</code>	156
6.1.2.3	<code>mux_select_t</code>	156
6.1.3	Function Documentation	157
6.1.3.1	<code>alt_mode_hw_deinit()</code>	157
6.1.3.2	<code>alt_mode_hw_is_idle()</code>	157
6.1.3.3	<code>alt_mode_hw_set_cbk()</code>	158
6.1.3.4	<code>alt_mode_hw_sleep()</code>	158
6.1.3.5	<code>alt_mode_hw_wakeup()</code>	158
6.1.3.6	<code>dp_snk_get_hpd_state()</code>	159
6.1.3.7	<code>eval_app_alt_hw_cmd()</code>	159
6.1.3.8	<code>eval_hpd_cmd()</code>	159
6.1.3.9	<code>eval_mux_cmd()</code>	160
6.1.3.10	<code>get_mux_state()</code>	160
6.1.3.11	<code>set_mux()</code>	160
6.2	app/alt_mode/alt_modes_mngr.h File Reference	161
6.2.1	Detailed Description	162
6.2.2	Typedef Documentation	162
6.2.2.1	<code>alt_mode_app_cbk_t</code>	162
6.2.2.2	<code>alt_mode_cbk_t</code>	163
6.2.3	Enumeration Type Documentation	163
6.2.3.1	<code>alt_mode_app_cmd_t</code>	163
6.2.3.2	<code>alt_mode_app_evt_t</code>	163
6.2.3.3	<code>alt_mode_mngr_state_t</code>	164
6.2.3.4	<code>alt_mode_state_t</code>	164
6.2.3.5	<code>fail_status_t</code>	165
6.2.4	Function Documentation	165
6.2.4.1	<code>alt_mode_get_status()</code>	165
6.2.4.2	<code>alt_mode_mngr_exit_all()</code>	165
6.2.4.3	<code>alt_mode_mngr_sleep()</code>	165
6.2.4.4	<code>alt_mode_mngr_wakeup()</code>	166
6.2.4.5	<code>eval_app_alt_mode_cmd()</code>	166
6.2.4.6	<code>eval_rec_vdm()</code>	167
6.2.4.7	<code>form_alt_mode_event()</code>	167
6.2.4.8	<code>get_alt_mode_numb()</code>	167
6.2.4.9	<code>get_vdm_buff()</code>	168
6.2.4.10	<code>is_alt_mode_mngr_idle()</code>	168
6.2.4.11	<code>is_svid_supported()</code>	168
6.2.4.12	<code>reg_alt_mode_mngr()</code>	169
6.2.4.13	<code>reset_alt_mode_info()</code>	169
6.2.4.14	<code>vdm_task_mngr_alt_mode_process()</code>	169

6.3	app/alt_mode/dp_sid.h File Reference	170
6.3.1	Detailed Description	171
6.3.2	Enumeration Type Documentation	171
6.3.2.1	dp_conn_t	171
6.3.2.2	dp_port_cap_t	171
6.3.2.3	dp_stat_bm_t	172
6.3.2.4	dp_state_t	172
6.3.3	Function Documentation	172
6.3.3.1	reg_dp_modes()	172
6.4	app/alt_mode/intel_ridge.h File Reference	173
6.4.1	Detailed Description	174
6.4.2	Function Documentation	174
6.4.2.1	ridge_eval_cmd()	174
6.4.2.2	ridge_set_ctrl_change_cb()	174
6.4.2.3	ridge_set_mux()	174
6.4.2.4	tr_hpd_deinit()	175
6.4.2.5	tr_hpd_init()	175
6.4.2.6	tr_hpd_sendevt()	176
6.4.2.7	tr_is_hpd_change()	176
6.5	app/alt_mode/intel_vid.h File Reference	176
6.5.1	Detailed Description	177
6.5.2	Enumeration Type Documentation	177
6.5.2.1	tbt_state_t	177
6.5.3	Function Documentation	177
6.5.3.1	reg_intel_modes()	177
6.6	app/alt_mode/vdm_task_mngr.h File Reference	178
6.6.1	Detailed Description	178
6.6.2	Enumeration Type Documentation	178
6.6.2.1	vdm_evt_t	178
6.6.2.2	vdm_task_t	179
6.6.3	Function Documentation	179
6.6.3.1	enable_vdm_task_mngr()	179
6.6.3.2	is_vdm_task_idle()	180
6.6.3.3	vdm_get_disc_id_resp()	180
6.6.3.4	vdm_get_disc_svid_resp()	180
6.6.3.5	vdm_task_mngr()	181
6.6.3.6	vdm_task_mngr_deinit()	181
6.7	app/app.h File Reference	181
6.7.1	Detailed Description	184
6.7.2	Typedef Documentation	184

6.7.2.1	<code>mux_poll_fnc_cbk_t</code>	184
6.7.3	Enumeration Type Documentation	184
6.7.3.1	<code>anonymous enum</code>	184
6.7.3.2	<code>anonymous enum</code>	184
6.7.3.3	<code>sys_hw_error_type_t</code>	185
6.7.4	Function Documentation	185
6.7.4.1	<code>app_conf_for_faulty_dev_removal()</code>	185
6.7.4.2	<code>app_disable_pd_port()</code>	185
6.7.4.3	<code>app_event_handler()</code>	186
6.7.4.4	<code>app_get_callback_ptr()</code>	186
6.7.4.5	<code>app_get_resp_buf()</code>	186
6.7.4.6	<code>app_get_status()</code>	187
6.7.4.7	<code>app_init()</code>	187
6.7.4.8	<code>app_ovp_disable()</code>	187
6.7.4.9	<code>app_ovp_enable()</code>	188
6.7.4.10	<code>app_sleep()</code>	188
6.7.4.11	<code>app_task()</code>	188
6.7.4.12	<code>app_update_bc_src_support()</code>	188
6.7.4.13	<code>app_update_sys_pwr_state()</code>	189
6.7.4.14	<code>app_uvp_disable()</code>	189
6.7.4.15	<code>app_uvp_enable()</code>	189
6.7.4.16	<code>app_validate_configurable_offsets()</code>	191
6.7.4.17	<code>app_wakeup()</code>	191
6.7.4.18	<code>ccg_app_task_init()</code>	191
6.7.4.19	<code>mux_ctrl_init()</code>	192
6.7.4.20	<code>mux_ctrl_set_cfg()</code>	192
6.7.4.21	<code>sln_pd_event_handler()</code>	192
6.7.4.22	<code>system_sleep()</code>	193
6.7.4.23	<code>system_vconn_ocp_dis()</code>	193
6.7.4.24	<code>system_vconn_ocp_en()</code>	193
6.7.4.25	<code>vbus_discharge_off()</code>	194
6.7.4.26	<code>vbus_discharge_on()</code>	194
6.7.4.27	<code>vbus_get_value()</code>	194
6.7.4.28	<code>vbus_is_present()</code>	195
6.7.4.29	<code>vconn_disable()</code>	195
6.7.4.30	<code>vconn_enable()</code>	195
6.7.4.31	<code>vconn_is_present()</code>	196
6.8	<code>app/billboard.h</code> File Reference	196
6.8.1	Detailed Description	197
6.8.2	Macro Definition Documentation	197

6.8.2.1	BB_MAX_EP0_XFER_SIZE	197
6.8.3	Enumeration Type Documentation	198
6.8.3.1	bb_alt_mode_status_t	198
6.8.3.2	bb_cause_t	198
6.8.3.3	bb_state_t	198
6.8.3.4	bb_type_t	199
6.8.3.5	bb_usb_string_index_t	199
6.8.4	Function Documentation	199
6.8.4.1	bb_bridge_ctrl()	199
6.8.4.2	bb_disable()	200
6.8.4.3	bb_enable()	200
6.8.4.4	bb_enter_deep_sleep()	201
6.8.4.5	bb_flashing_ctrl()	201
6.8.4.6	bb_get_version()	201
6.8.4.7	bb_init()	202
6.8.4.8	bb_is_idle()	202
6.8.4.9	bb_is_present()	202
6.8.4.10	bb_task()	203
6.8.4.11	bb_update_all_status()	203
6.8.4.12	bb_update_alt_status()	204
6.9	app/pdo.h File Reference	204
6.9.1	Detailed Description	204
6.9.2	Function Documentation	204
6.9.2.1	eval_rdo()	204
6.9.2.2	eval_src_cap()	205
6.10	app/psink.h File Reference	205
6.10.1	Detailed Description	205
6.10.2	Function Documentation	205
6.10.2.1	psnk_disable()	205
6.10.2.2	psnk_enable()	206
6.10.2.3	psnk_set_current()	206
6.10.2.4	psnk_set_voltage()	206
6.11	app/psource.h File Reference	207
6.11.1	Detailed Description	207
6.11.2	Function Documentation	207
6.11.2.1	psrc_disable()	207
6.11.2.2	psrc_enable()	208
6.11.2.3	psrc_get_voltage()	208
6.11.2.4	psrc_set_current()	208
6.11.2.5	psrc_set_voltage()	209

6.12 app/ridge_slave/ridge_slave.h File Reference	209
6.12.1 Detailed Description	210
6.12.2 Macro Definition Documentation	210
6.12.2.1 RIDGE_SLAVE_ADDR_MASK	210
6.12.3 Enumeration Type Documentation	210
6.12.3.1 ridge_slave_reg_addr_t	210
6.12.4 Function Documentation	210
6.12.4.1 ridge_reg_reset()	211
6.12.4.2 ridge_slave_init()	211
6.12.4.3 ridge_slave_is_host_connected()	211
6.12.4.4 ridge_slave_sleep()	212
6.12.4.5 ridge_slave_status_update()	212
6.12.4.6 ridge_slave_task()	212
6.12.4.7 ridge_slave_update_ocp_status()	213
6.13 app/swap.h File Reference	213
6.13.1 Detailed Description	213
6.13.2 Function Documentation	213
6.13.2.1 eval_dr_swap()	213
6.13.2.2 eval_pr_swap()	214
6.13.2.3 eval_vconn_swap()	214
6.14 app/usb_hid.h File Reference	214
6.14.1 Detailed Description	215
6.14.2 Enumeration Type Documentation	215
6.14.2.1 hid_report_id_t	215
6.14.2.2 hid_rqt_cmd_t	216
6.14.3 Function Documentation	217
6.14.3.1 usb_hid_class_rqt_handler()	217
6.14.3.2 usb_hid_flashing_rqt()	217
6.14.3.3 usb_hid_get_ep0_buffer()	217
6.14.3.4 usb_hid_get_inf_dscr()	218
6.14.3.5 usb_hid_get_report_dscr()	218
6.14.3.6 usb_hid_inf_ctrl()	219
6.14.3.7 usb_hid_set_fw_locations()	219
6.14.3.8 usb_i2cm_bridge_ctrl()	219
6.15 app/usb_i2cm.h File Reference	220
6.15.1 Detailed Description	221
6.15.2 Enumeration Type Documentation	221
6.15.2.1 usb_i2cm_vdr_rqt_t	221
6.15.3 Function Documentation	221
6.15.3.1 usb_i2cm_get_dscr_rqt_handler()	221

6.15.3.2 <code>usb_i2cm_get_ep0_buffer()</code>	222
6.15.3.3 <code>usb_i2cm_inf_ctrl()</code>	222
6.15.3.4 <code>usb_i2cm_is_idle()</code>	222
6.15.3.5 <code>usb_i2cm_task()</code>	223
6.15.3.6 <code>usb_i2cm_vendor_rqt_handler()</code>	223
6.16 <code>app/uvdm.h</code> File Reference	223
6.16.1 Detailed Description	225
6.16.2 Enumeration Type Documentation	225
6.16.2.1 <code>uvdm_cmd_opcode_t</code>	225
6.16.2.2 <code>uvdm_response_state_t</code>	225
6.17 <code>app/vdm.h</code> File Reference	226
6.17.1 Detailed Description	226
6.17.2 Function Documentation	226
6.17.2.1 <code>eval_vdm()</code>	226
6.17.2.2 <code>get_modes_vdo_info()</code>	227
6.17.2.3 <code>vdm_data_init()</code>	227
6.17.2.4 <code>vdm_update_data()</code>	227
6.17.2.5 <code>vdm_update_svid_resp()</code>	228
6.18 <code>hpiss/hpi.h</code> File Reference	228
6.18.1 Detailed Description	230
6.18.2 Typedef Documentation	230
6.18.2.1 <code>hpi_write_cb_t</code>	230
6.18.3 Enumeration Type Documentation	230
6.18.3.1 <code>hpi_boot_prio_conf_t</code>	230
6.18.3.2 <code>hpi_reg_part_t</code>	230
6.18.3.3 <code>hpi_reg_section_t</code>	231
6.18.4 Function Documentation	231
6.18.4.1 <code>get_hpi_sof_reset_timer_id()</code>	231
6.18.4.2 <code>get_hpi_soft_reset_delay()</code>	231
6.18.4.3 <code>hpi_deinit()</code>	232
6.18.4.4 <code>hpi_get_port_enable()</code>	232
6.18.4.5 <code>hpi_get_sys_pwr_state()</code>	232
6.18.4.6 <code>hpi_init()</code>	232
6.18.4.7 <code>hpi_init_userdef_regs()</code>	233
6.18.4.8 <code>hpi_is_accessed()</code>	233
6.18.4.9 <code>hpi_is_ec_ready()</code>	233
6.18.4.10 <code>hpi_is_extd_msg_ec_ctrl_enabled()</code>	234
6.18.4.11 <code>hpi_is_vdm_ec_ctrl_enabled()</code>	234
6.18.4.12 <code>hpi_pd_event_handler()</code>	234
6.18.4.13 <code>hpi_reg_enqueue_event()</code>	235

6.18.4.14 hpi_send_fw_ready_event()	235
6.18.4.15 hpi_send_hw_error_event()	235
6.18.4.16 hpi_set_boot_priority_conf()	236
6.18.4.17 hpi_set_fixed_slave_address()	236
6.18.4.18 hpi_set_flash_params()	236
6.18.4.19 hpi_set_mode_regs()	237
6.18.4.20 hpi_set_no_boot_mode()	237
6.18.4.21 hpi_set_port_event_mask()	237
6.18.4.22 hpi_set_reserved_reg_35()	238
6.18.4.23 hpi_set_reserved_reg_37()	238
6.18.4.24 hpi_set_reset_count()	238
6.18.4.25 hpi_set_userdef_write_handler()	240
6.18.4.26 hpi_sleep()	240
6.18.4.27 hpi_sleep_allowed()	240
6.18.4.28 hpi_task()	241
6.18.4.29 hpi_update_fw_locations()	241
6.18.4.30 hpi_update_pdo_change()	241
6.18.4.31 hpi_update_versions()	242
6.18.4.32 hpid_get_ec_active_modes()	242
6.18.4.33 update_hpi_sof_reset_timer_id()	242
6.18.4.34 update_hpi_soft_reset_delay()	243
6.19 pd_common/dpm.h File Reference	243
6.19.1 Detailed Description	244
6.19.2 Function Documentation	244
6.19.2.1 dpm_clear_fault_active()	244
6.19.2.2 dpm_clear_hard_reset_count()	244
6.19.2.3 dpm_deepsleep()	245
6.19.2.4 dpm_disable()	245
6.19.2.5 dpm_get_def_cable_cap()	245
6.19.2.6 dpm_get_evtno_n_clear()	246
6.19.2.7 dpm_get_info()	246
6.19.2.8 dpm_get_mux_enable_wait_period()	246
6.19.2.9 dpm_get_pd_port_status()	247
6.19.2.10 dpm_get_polarity()	247
6.19.2.11 dpm_get_sink_detach_margin()	247
6.19.2.12 dpm_get_sink_detach_voltage()	247
6.19.2.13 dpm_get_snk_wait_cap_period()	248
6.19.2.14 dpm_init()	248
6.19.2.15 dpm_is_rdo_valid()	248
6.19.2.16 dpm_pd_command()	249

6.19.2.17 dpm_pd_command_ec()	249
6.19.2.18 dpm_pe_stop()	250
6.19.2.19 dpm_prot_reset()	250
6.19.2.20 dpm_prot_reset_rx()	251
6.19.2.21 dpm_send_hard_reset()	251
6.19.2.22 dpm_set_alert()	251
6.19.2.23 dpm_set_cf()	252
6.19.2.24 dpm_set_chunk_transfer_running()	252
6.19.2.25 dpm_set_fault_active()	252
6.19.2.26 dpm_sleep()	253
6.19.2.27 dpm_start()	253
6.19.2.28 dpm_stop()	253
6.19.2.29 dpm_task()	254
6.19.2.30 dpm_typec_command()	254
6.19.2.31 dpm_typec_deassert_rp_rd()	254
6.19.2.32 dpm_update_def_cable_cap()	255
6.19.2.33 dpm_update_ext_src_cap()	255
6.19.2.34 dpm_update_frs_enable()	255
6.19.2.35 dpm_update_mux_enable_wait_period()	256
6.19.2.36 dpm_update_port_config()	256
6.19.2.37 dpm_update_port_status()	257
6.19.2.38 dpm_update_snk_cap()	257
6.19.2.39 dpm_update_snk_cap_mask()	257
6.19.2.40 dpm_update_snk_max_min()	258
6.19.2.41 dpm_update_snk_wait_cap_period()	258
6.19.2.42 dpm_update_src_cap()	259
6.19.2.43 dpm_update_src_cap_mask()	259
6.19.2.44 dpm_update_swap_response()	259
6.19.2.45 dpm_wakeup()	260
6.20 pd_common/pd.h File Reference	260
6.20.1 Detailed Description	267
6.20.2 Macro Definition Documentation	268
6.20.2.1 BC_SINK_1_2_MODE_ENABLE_MASK	268
6.20.2.2 BC_SINK_APPLE_MODE_ENABLE_MASK	268
6.20.2.3 BC_SRC_1_2_MODE_ENABLE_MASK	268
6.20.2.4 BC_SRC_AFC_MODE_ENABLE_MASK	268
6.20.2.5 BC_SRC_APPLE_MODE_ENABLE_MASK	268
6.20.2.6 BC_SRC_QC_4_0_MODE_ENABLE_MASK	268
6.20.2.7 BC_SRC_QC_MODE_ENABLE_MASK	268
6.20.2.8 BC_SRC_QC_VER_2_CLASS_A_VAL	268

6.20.2.9 BC_SRC_QC_VER_2_CLASS_B_VAL	269
6.20.2.10 BC_SRC_QC_VER_3_CLASS_A_VAL	269
6.20.2.11 BC_SRC_QC_VER_3_CLASS_B_VAL	269
6.20.2.12 BDO_HDR_IDX	269
6.20.2.13 CC_CHANNEL_1	269
6.20.2.14 CC_CHANNEL_2	269
6.20.2.15 CCG_CC_STAT_DRP_TOGGLE	269
6.20.2.16 CCG_CC_STAT_RD_PRESENT	269
6.20.2.17 CCG_CC_STAT_RP_PRESENT	269
6.20.2.18 CCG_CC_STAT_VCONN_ACTIVE	270
6.20.2.19 CCG_CC_STAT_ZOPEN	270
6.20.2.20 CCG_DPM_ERROR_NO_VCONN	270
6.20.2.21 CCG_DPM_ERROR_NONE	270
6.20.2.22 CCG_FRS_RX_ENABLE_MASK	270
6.20.2.23 CCG_FRS_TX_ENABLE_MASK	270
6.20.2.24 CCG_PD_EXT_PPS_STATUS_SIZE	270
6.20.2.25 CCG_PD_EXT_SRCCAP_INP_INDEX	270
6.20.2.26 CCG_PD_EXT_SRCCAP_INP_UNCONSTRAINED	270
6.20.2.27 CCG_PD_EXT_SRCCAP_PDP_INDEX	271
6.20.2.28 CCG_PD_EXT_SRCCAP_SIZE	271
6.20.2.29 CCG_PD_EXT_STATUS_SIZE	271
6.20.2.30 CCG_PD_FIX_SRC_PDO_MASK_REV2	271
6.20.2.31 CCG_PD_FIX_SRC_PDO_MASK_REV3	271
6.20.2.32 CCG_PD_FLAG_CONTRACT_NEG_ACTIVE	271
6.20.2.33 CCG_PD_FLAG_EXPLICIT_CONTRACT	271
6.20.2.34 CCG_PD_FLAG_POWER_SINK	271
6.20.2.35 CCG_PD_FLAG_SRC_READY	271
6.20.2.36 CY_VID	272
6.20.2.37 DP_SVID	272
6.20.2.38 DRP_TOGGLE_PERIOD	272
6.20.2.39 GET_DR_SWAP_RESP	272
6.20.2.40 GET_PR_SWAP_RESP	272
6.20.2.41 GET_VCONN_SWAP_RESP	272
6.20.2.42 GIVE_BACK_MASK	272
6.20.2.43 HPD_RX_ACTIVITY_TIMER_PERIOD_MAX	272
6.20.2.44 HPD_RX_ACTIVITY_TIMER_PERIOD_MIN	273
6.20.2.45 I_1A	273
6.20.2.46 I_1P5A	273
6.20.2.47 I_2A	273
6.20.2.48 I_3A	273

6.20.2.49 I_5A	273
6.20.2.50 ID_HEADER_IDX	273
6.20.2.51 ISAFE_0A	273
6.20.2.52 ISAFE_DEF	273
6.20.2.53 MAX_CBL_DSC_ID_COUNT	274
6.20.2.54 MAX_EXTD_MSG_LEGACY_LEN	274
6.20.2.55 MAX_EXTD_PKT_SIZE	274
6.20.2.56 MAX_EXTD_PKT_WORDS	274
6.20.2.57 MAX_HARD_RESET_COUNT	274
6.20.2.58 MAX_MESSAGE_ID	274
6.20.2.59 MAX_NO_OF_DO	274
6.20.2.60 MAX_NO_OF_PDO	274
6.20.2.61 MAX_NO_OF_VDO	274
6.20.2.62 MAX_SOP_TYPES	275
6.20.2.63 MAX_SRC_CAP_COUNT	275
6.20.2.64 PD_BIST_CONT_MODE_TIMER_PERIOD	275
6.20.2.65 PD_CBL_DELAY_TIMER_PERIOD	275
6.20.2.66 PD_CBL_DSC_ID_START_TIMER_PERIOD	275
6.20.2.67 PD_CBL_DSC_ID_TIMER_PERIOD	275
6.20.2.68 PD_CBL_READY_TIMER_PERIOD	275
6.20.2.69 PD_COLLISION_SRC_COOL_OFF_TIMER_PERIOD	275
6.20.2.70 PD_CUR_PER_UNIT	275
6.20.2.71 PD_DPM_RESP_REC_RESP_PERIOD	276
6.20.2.72 PD_EXTERNALLY_POWERED_BIT_POS	276
6.20.2.73 PD_HARD_RESET_TX_TIMER_PERIOD	276
6.20.2.74 PD_MAX_SRC_CAP_TRY	276
6.20.2.75 PD_NO_RESPONSE_TIMER_PERIOD	276
6.20.2.76 PD_PHY_BUSY_TIMER_PERIOD	276
6.20.2.77 PD_PPS_SRC_TIMER_PERIOD	276
6.20.2.78 PD_PS_HARD_RESET_TIMER_PERIOD	276
6.20.2.79 PD_PS_SNK_TRANSITION_TIMER_PERIOD	276
6.20.2.80 PD_PS_SRC_OFF_TIMER_PERIOD	277
6.20.2.81 PD_PS_SRC_ON_TIMER_PERIOD	277
6.20.2.82 PD_PS_SRC_TRANS_TIMER_PERIOD	277
6.20.2.83 PD_RECEIVER_RESPONSE_TIMER_PERIOD	277
6.20.2.84 PD_SENDER_RESPONSE_TIMER_PERIOD	277
6.20.2.85 PD_SINK_TX_TIMER_PERIOD	277
6.20.2.86 PD_SINK_VBUS_TURN_OFF_TIMER_PERIOD	277
6.20.2.87 PD_SINK_VBUS_TURN_ON_TIMER_PERIOD	277
6.20.2.88 PD_SINK_WAIT_CAP_TIMER_PERIOD	277

6.20.2.89 PD_SOURCE_TRANSITION_TIMER_PERIOD	278
6.20.2.90 PD_SRC_CAP_TIMER_PERIOD	278
6.20.2.91 PD_SRC_RECOVER_TIMER_PERIOD	278
6.20.2.92 PD_SWAP_SRC_START_TIMER_PERIOD	278
6.20.2.93 PD_VBUS_TURN_OFF_TIMER_PERIOD	278
6.20.2.94 PD_VBUS_TURN_ON_TIMER_PERIOD	278
6.20.2.95 PD_VCONN_OFF_TIMER_PERIOD	278
6.20.2.96 PD_VCONN_ON_TIMER_PERIOD	278
6.20.2.97 PD_VCONN_SWAP_INITIATOR_TIMER_PERIOD	278
6.20.2.98 PD_VCONN_TURN_ON_TIMER_PERIOD	279
6.20.2.99 PD_VDM_ENTER_MODE_RESPONSE_TIMER_PERIOD	279
6.20.2.100PD_VDM_EXIT_MODE_RESPONSE_TIMER_PERIOD	279
6.20.2.101PD_VDM_RESPONSE_TIMER_PERIOD	279
6.20.2.102PD_VOLT_PER_UNIT	279
6.20.2.103RDO_IDX	279
6.20.2.104SNK_DETACH_VBUS_POLL_COUNT	279
6.20.2.105SNK_MIN_MAX_MASK	279
6.20.2.106SRC_DRP_MIN_DC	279
6.20.2.107STD_SVID	280
6.20.2.108STD_VDM_VERSION	280
6.20.2.109STD_VDM_VERSION_IDX	280
6.20.2.110STD_VDM_VERSION_REV2	280
6.20.2.111STD_VDM_VERSION_REV3	280
6.20.2.112TBT_SVID	280
6.20.2.113TYPEC_CC_DEBOUNCE_TIMER_PERIOD	280
6.20.2.114TYPEC_DRP_TRY_TIMER_PERIOD	280
6.20.2.115TYPEC_ERROR_RECOVERY_TIMER_PERIOD	280
6.20.2.116TYPEC_FSM_GENERIC	281
6.20.2.117TYPEC_FSM_NONE	281
6.20.2.118TYPEC_PD3_RPCHANGE_DEBOUNCE_PERIOD	281
6.20.2.119TYPEC_PD_DEBOUNCE_TIMER_PERIOD	281
6.20.2.120TYPEC_RD_DEBOUNCE_TIMER_PERIOD	281
6.20.2.121TYPEC_SRC_DETACH_DEBOUNCE_PERIOD	281
6.20.2.122VDM_HEADER_IDX	281
6.20.2.123VSAFE_0V	281
6.20.2.124VSAFE_0V_HARD_RESET	281
6.20.2.125VSAFE_0V_PR_SWAP_SNK_SRC	282
6.20.2.126VSAFE_12V	282
6.20.2.127VSAFE_13V	282
6.20.2.128VSAFE_15V	282

6.20.2.129VSAFE_19V	282
6.20.2.130VSAFE_20V	282
6.20.2.131VSAFE_3_6V	282
6.20.2.132VSAFE_5V	282
6.20.2.133VSAFE_9V	282
6.20.3 Typedef Documentation	283
6.20.3.1 app_resp_cbk_t	283
6.20.3.2 dpm_pd_cmd_cbk_t	283
6.20.3.3 dpm_typec_cmd_cbk_t	283
6.20.3.4 pd_cbk_t	283
6.20.3.5 pwr_ready_cbk_t	284
6.20.3.6 sink_discharge_off_cbk_t	284
6.20.3.7 vdm_resp_cbk_t	284
6.20.4 Enumeration Type Documentation	284
6.20.4.1 apdo_t	284
6.20.4.2 app_evt_t	285
6.20.4.3 app_fault_mask_t	286
6.20.4.4 app_req_status_t	286
6.20.4.5 app_swap_resp_t	287
6.20.4.6 bist_mode_t	287
6.20.4.7 cbl_term_t	287
6.20.4.8 cbl_vbus_cur_t	288
6.20.4.9 ctrl_msg_t	288
6.20.4.10 data_msg_t	288
6.20.4.11 dpm_pd_cmd_t	289
6.20.4.12 dpm_typec_cmd_resp_t	289
6.20.4.13 dpm_typec_cmd_t	290
6.20.4.14 extd_msg_t	290
6.20.4.15 fr_swap_supp_t	291
6.20.4.16 pd_ams_type	291
6.20.4.17 pd_cable_reset_reason_t	291
6.20.4.18 pd_contract_status_t	291
6.20.4.19 pd_devtype_t	292
6.20.4.20 pd_emca_sr_reason_t	292
6.20.4.21 pd_err_recov_reason_t	293
6.20.4.22 pd_hard_reset_reason_t	293
6.20.4.23 pd_msg_class_t	293
6.20.4.24 pd_rev_t	294
6.20.4.25 pd_soft_reset_reason_t	294
6.20.4.26 pd_timer_id_t	294

6.20.4.27 pdo_t	295
6.20.4.28 pe_cbl_state_t	295
6.20.4.29 pe_fsm_state_t	296
6.20.4.30 peak_cur_cap_t	297
6.20.4.31 port_role_t	297
6.20.4.32 port_type_t	297
6.20.4.33 rd_cc_status_t	298
6.20.4.34 rdo_type_t	298
6.20.4.35 resp_status_t	298
6.20.4.36 rp_cc_status_t	298
6.20.4.37 rp_term_t	299
6.20.4.38 sop_t	299
6.20.4.39 std_vdm_cmd_t	299
6.20.4.40 std_vdm_cmd_type_t	300
6.20.4.41 std_vdm_prod_t	300
6.20.4.42 std_vdm_ver_t	300
6.20.4.43 try_src_snk_t	301
6.20.4.44 typec_fsm_state_t	301
6.20.4.45 vdm_ams_t	302
6.20.4.46 vdm_type_t	302
6.20.5 Function Documentation	302
6.20.5.1 get_pd_config()	302
6.20.5.2 get_pd_port_config()	302
6.20.5.3 pd_get_ptr_bat_chg_tbl()	303
6.20.5.4 pd_get_ptr_bb_tbl()	303
6.20.5.5 pd_get_ptr_chg_cfg_tbl()	304
6.20.5.6 pd_get_ptr_ocp_tbl()	304
6.20.5.7 pd_get_ptr_otp_tbl()	304
6.20.5.8 pd_get_ptr_ovp_tbl()	305
6.20.5.9 pd_get_ptr_pwr_tbl()	305
6.20.5.10 pd_get_ptr_scp_tbl()	305
6.20.5.11 pd_get_ptr_type_a_chg_cfg_tbl()	306
6.20.5.12 pd_get_ptr_type_a_pwr_tbl()	306
6.20.5.13 pd_get_ptr_uvp_tbl()	306
6.20.5.14 pd_get_ptr_vconn_ocp_tbl()	307
6.20.5.15 pd_is_msg()	307
6.21 pd_common/pd_policy_engine.h File Reference	307
6.21.1 Detailed Description	308
6.21.2 Function Documentation	308
6.21.2.1 get_pe_state_buf()	308

6.21.2.2 <code>get_spec_rev_determined()</code>	308
6.21.2.3 <code>pe_clear_hard_reset_count()</code>	309
6.21.2.4 <code>pe_disabled()</code>	309
6.21.2.5 <code>pe_fsm()</code>	309
6.21.2.6 <code>pe_get_pps_status()</code>	309
6.21.2.7 <code>pe_init()</code>	310
6.21.2.8 <code>pe_is_busy()</code>	310
6.21.2.9 <code>pe_push_to_buf()</code>	310
6.21.2.10 <code>pe_start()</code>	310
6.21.2.11 <code>pe_stop()</code>	311
6.22 <code>pd_common/pd_protocol.h</code> File Reference	311
6.22.1 Detailed Description	311
6.22.2 Function Documentation	312
6.22.2.1 <code>pd_prot_dis_bist_cm2()</code>	312
6.22.2.2 <code>pd_prot_dis_bist_test_data()</code>	312
6.22.2.3 <code>pd_prot_en_bist_cm2()</code>	312
6.22.2.4 <code>pd_prot_en_bist_test_data()</code>	313
6.22.2.5 <code>pd_prot_get_rx_packet()</code>	313
6.22.2.6 <code>pd_prot_init()</code>	313
6.22.2.7 <code>pd_prot_is_busy()</code>	314
6.22.2.8 <code>pd_prot_refresh_roles()</code>	314
6.22.2.9 <code>pd_prot_reset()</code>	314
6.22.2.10 <code>pd_prot_reset_all()</code>	315
6.22.2.11 <code>pd_prot_reset_rx()</code>	315
6.22.2.12 <code>pd_prot_rx_dis()</code>	315
6.22.2.13 <code>pd_prot_rx_en()</code>	316
6.22.2.14 <code>pd_prot_send_cable_reset()</code>	316
6.22.2.15 <code>pd_prot_send_ctrl_msg()</code>	316
6.22.2.16 <code>pd_prot_send_data_msg()</code>	317
6.22.2.17 <code>pd_prot_send_hard_reset()</code>	317
6.22.2.18 <code>pd_prot_set_avoid_retry()</code>	317
6.22.2.19 <code>pd_prot_start()</code>	318
6.22.2.20 <code>pd_prot_stop()</code>	318
6.23 <code>pd_common/typec_manager.h</code> File Reference	318
6.23.1 Detailed Description	319
6.23.2 Function Documentation	319
6.23.2.1 <code>typec_assert_rd()</code>	319
6.23.2.2 <code>typec_assert_rp()</code>	319
6.23.2.3 <code>typec_change_rp()</code>	320
6.23.2.4 <code>typec_deepsleep()</code>	320

6.23.2.5 typec_fsm()	320
6.23.2.6 typec_init()	320
6.23.2.7 typec_is_busy()	321
6.23.2.8 typec_start()	321
6.23.2.9 typec_stop()	321
6.23.2.10 typec_wakeup()	321
6.24 pd_hal/hal_ccgx.h File Reference	322
6.24.1 Detailed Description	322
6.24.2 Macro Definition Documentation	322
6.24.2.1 CCG_SILICON_REV00_VALUE	322
6.24.3 Function Documentation	322
6.24.3.1 ccg_get_si_revision()	323
6.24.3.2 system_connect_ovp_trip()	323
6.24.3.3 system_disconnect_ovp_trip()	323
6.24.3.4 system_init()	323
6.24.3.5 system_vbus_ocp_dis()	324
6.24.3.6 system_vbus_ocp_en()	324
6.24.3.7 system_vbus_scp_dis()	324
6.24.3.8 system_vbus_scp_en()	325
6.24.3.9 vbus_ocp_handler()	325
6.24.3.10 vbus_scp_handler()	325
6.25 pd_hal/hpd.h File Reference	326
6.25.1 Detailed Description	326
6.25.2 Enumeration Type Documentation	327
6.25.2.1 hpd_event_type_t	327
6.25.3 Function Documentation	327
6.25.3.1 hpd_deinit()	327
6.25.3.2 hpd_receive_get_status()	327
6.25.3.3 hpd_receive_init()	328
6.25.3.4 hpd_rx_sleep_entry()	328
6.25.3.5 hpd_rx_wakeup()	328
6.25.3.6 hpd_sleep_entry()	329
6.25.3.7 hpd_transmit_init()	329
6.25.3.8 hpd_transmit_sendevt()	329
6.25.3.9 hpd_wakeup()	330
6.25.3.10 is_hpd_rx_state_idle()	330
6.26 pd_hal/pdss_hal.h File Reference	330
6.26.1 Detailed Description	335
6.26.2 Macro Definition Documentation	335
6.26.2.1 FRS_TX_SWAP_CTRL1_DFLT_VAL	335

6.26.2.2	HEADER_INFO_CFG	335
6.26.2.3	RCV_INTR_MASK	336
6.26.2.4	RST_TX_INTERRUPTS	336
6.26.2.5	RX_CC_CFG	336
6.26.2.6	RX_INTERRUPTS	336
6.26.2.7	RX_ORDER_SET_CTRL_CFG	336
6.26.2.8	TX_INTERRUPTS	337
6.26.2.9	VBUS_OCP_GPIO_ACTIVE_HIGH	337
6.26.2.10	VBUS_OCP_GPIO_ACTIVE_LOW	337
6.26.2.11	VBUS_OCP_MODE_EXT	337
6.26.2.12	VBUS_OCP_MODE_INT	337
6.26.2.13	VBUS_OCP_MODE_INT_AUTOCTRL	337
6.26.2.14	VBUS_OCP_MODE_INT_SW_DB	337
6.26.2.15	VBUS_OCP_MODE_POLLING	337
6.26.3	Typedef Documentation	338
6.26.3.1	PD_ADC_CB_T	338
6.26.3.2	pd_cmp_cbk_t	338
6.26.3.3	pd_phy_cbk_t	338
6.26.3.4	pd_supply_change_cbk_t	338
6.26.3.5	vbus_cf_cbk_t	339
6.26.4	Enumeration Type Documentation	339
6.26.4.1	aux_resistor_config_t	339
6.26.4.2	ccg_supply_t	339
6.26.4.3	comp_id_t	340
6.26.4.4	comp_tr_id_t	340
6.26.4.5	dpdm_mux_cfg_t	340
6.26.4.6	filter_edge_detect_cfg_t	341
6.26.4.7	filter_id_t	341
6.26.4.8	lscsa_app_config_t	341
6.26.4.9	PD_ADC_ID_T	342
6.26.4.10	PD_ADC_INPUT_T	342
6.26.4.11	PD_ADC_INT_T	342
6.26.4.12	pd_phy_evt_t	343
6.26.4.13	sbu_switch_state_t	343
6.26.4.14	vbus_ovp_mode_t	343
6.26.4.15	vbus_upv_mode_t	344
6.26.5	Function Documentation	344
6.26.5.1	pd_adc_calibrate()	344
6.26.5.2	pd_adc_comparator_ctrl()	345
6.26.5.3	pd_adc_comparator_sample()	345

6.26.5.4 pd_adc_free_run_ctrl()	346
6.26.5.5 pd_adc_get_comparator_status()	346
6.26.5.6 pd_adc_get_vbus_voltage()	347
6.26.5.7 pd_adc_init()	347
6.26.5.8 pd_adc_level_to_volt()	347
6.26.5.9 pd_adc_sample()	348
6.26.5.10 pd_adc_volt_to_level()	348
6.26.5.11 pd_cf_disable()	349
6.26.5.12 pd_cf_enable()	349
6.26.5.13 pd_cf_get_status()	349
6.26.5.14 pd_cmp_get_status()	350
6.26.5.15 pd_get_vbus_adc_level()	350
6.26.5.16 pd_hal_abort_auto_toggle()	351
6.26.5.17 pd_hal_cleanup()	351
6.26.5.18 pd_hal_config_auto_toggle()	351
6.26.5.19 pd_hal_dual_fet_config()	352
6.26.5.20 pd_hal_get_vbus_detach_adc()	352
6.26.5.21 pd_hal_get_vbus_detach_input()	352
6.26.5.22 pd_hal_init()	353
6.26.5.23 pd_hal_is_auto_toggle_active()	353
6.26.5.24 pd_hal_measure_vbus()	353
6.26.5.25 pd_hal_set_fet_drive()	354
6.26.5.26 pd_hal_set_supply_change_evt_cb()	354
6.26.5.27 pd_hal_set_vbus_detach_params()	355
6.26.5.28 pd_hal_set_vbus_mon_divider()	355
6.26.5.29 pd_hal_typec_sm_restart()	355
6.26.5.30 pd_hal_vconn_ocp_disable()	356
6.26.5.31 pd_hal_vconn_ocp_enable()	356
6.26.5.32 pd_is_vconn_present()	356
6.26.5.33 pd_lscsa_calc_cfg()	357
6.26.5.34 pd_lscsa_cfg()	357
6.26.5.35 pd_phy_abort_bist_cm2()	357
6.26.5.36 pd_phy_abort_tx_msg()	358
6.26.5.37 pd_phy_deepsleep()	358
6.26.5.38 pd_phy_dis_unchunked_tx()	358
6.26.5.39 pd_phy_en_unchunked_tx()	358
6.26.5.40 pd_phy_get_rx_packet()	359
6.26.5.41 pd_phy_init()	359
6.26.5.42 pd_phy_is_busy()	359
6.26.5.43 pd_phy_load_msg()	361

6.26.5.44 pd_phy_refresh_roles()	361
6.26.5.45 pd_phy_reset_rx_tx_sm()	362
6.26.5.46 pd_phy_send_bist_cm2()	362
6.26.5.47 pd_phy_send_msg()	362
6.26.5.48 pd_phy_send_reset()	363
6.26.5.49 pd_phy_wakeup()	363
6.26.5.50 pd_set_pfc_comp()	363
6.26.5.51 pd_set_sr_comp()	364
6.26.5.52 pd_stop_pfc_comp()	364
6.26.5.53 pd_stop_sr_comp()	364
6.26.5.54 pd_typec_dis_dpslp_rp()	365
6.26.5.55 pd_typec_dis_rd()	365
6.26.5.56 pd_typec_dis_rp()	365
6.26.5.57 pd_typec_en_dpslp_rp()	366
6.26.5.58 pd_typec_en_rd()	366
6.26.5.59 pd_typec_en_rp()	366
6.26.5.60 pd_typec_get_cc_status()	367
6.26.5.61 pd_typec_init()	367
6.26.5.62 pd_typec_rd_enable()	367
6.26.5.63 pd_typec_set_polarity()	368
6.26.5.64 pd_typec_snk_update_trim()	368
6.26.5.65 pd_typec_start()	368
6.26.5.66 pd_typec_stop()	369
6.26.5.67 pd_vconn_disable()	369
6.26.5.68 pd_vconn_enable()	369
6.26.5.69 soln_no_vsys_handler()	370
6.26.5.70 soln_vsys_removal_handler()	370
6.27 scb/i2c.h File Reference	370
6.27.1 Detailed Description	371
6.27.2 Macro Definition Documentation	371
6.27.2.1 I2C_BLOCK_COUNT	371
6.27.3 Enumeration Type Documentation	371
6.27.3.1 i2c_cb_cmd_t	371
6.27.3.2 i2c_scb_clock_freq_t	372
6.27.3.3 i2c_scb_mode_t	372
6.27.3.4 i2c_scb_state_t	372
6.27.4 Function Documentation	373
6.27.4.1 i2c_reset()	373
6.27.4.2 i2c_scb_deinit()	373
6.27.4.3 i2c_scb_enable_wakeup()	374

6.27.4.4 <code>i2c_scb_init()</code>	374
6.27.4.5 <code>i2c_scb_is_idle()</code>	375
6.27.4.6 <code>i2c_scb_write()</code>	375
6.27.4.7 <code>i2c_slave_ack_ctrl()</code>	375
6.28 <code>system/boot.h</code> File Reference	376
6.28.1 Detailed Description	377
6.28.2 Function Documentation	377
6.28.2.1 <code>boot_check_for_valid_fw()</code>	377
6.28.2.2 <code>boot_get_boot_seq()</code>	377
6.28.2.3 <code>boot_get_wait_time()</code>	378
6.28.2.4 <code>boot_handle_validate_fw_cmd()</code>	378
6.28.2.5 <code>boot_jump_to_fw()</code>	378
6.28.2.6 <code>boot_start()</code>	378
6.28.2.7 <code>boot_update_fw_status()</code>	379
6.28.2.8 <code>boot_validate_configtable()</code>	379
6.28.2.9 <code>boot_validate_fw()</code>	379
6.28.2.10 <code>get_boot_mode_reason()</code>	380
6.29 <code>system/ccgx_api_desc.h</code> File Reference	380
6.29.1 Detailed Description	380
6.30 <code>system/ccgx_version.h</code> File Reference	380
6.30.1 Detailed Description	380
6.30.2 Macro Definition Documentation	380
6.30.2.1 <code>FW_BASE_VERSION</code>	381
6.31 <code>system/flash.h</code> File Reference	381
6.31.1 Detailed Description	381
6.31.2 Typedef Documentation	382
6.31.2.1 <code>flash_cbk_t</code>	382
6.31.3 Enumeration Type Documentation	382
6.31.3.1 <code>flash_app_priority_t</code>	382
6.31.3.2 <code>flash_interface_t</code>	382
6.31.3.3 <code>flash_write_status_t</code>	383
6.31.4 Function Documentation	383
6.31.4.1 <code>flash_access_get_status()</code>	383
6.31.4.2 <code>flash_enter_mode()</code>	383
6.31.4.3 <code>flash_row_clear()</code>	384
6.31.4.4 <code>flash_row_read()</code>	384
6.31.4.5 <code>flash_row_write()</code>	385
6.31.4.6 <code>flash_set_access_limits()</code>	385
6.31.4.7 <code>flash_set_app_priority()</code>	386
6.32 <code>system/gpio.h</code> File Reference	386

6.32.1	Detailed Description	387
6.32.2	Enumeration Type Documentation	387
6.32.2.1	gpio_dm_t	387
6.32.2.2	gpio_intr_t	388
6.32.2.3	gpio_port_pin_t	388
6.32.2.4	hsiom_mode_t	390
6.32.3	Function Documentation	391
6.32.3.1	gpio_clear_intr()	391
6.32.3.2	gpio_get_intr()	391
6.32.3.3	gpio_hsiom_set_config()	391
6.32.3.4	gpio_int_set_config()	392
6.32.3.5	gpio_read_value()	392
6.32.3.6	gpio_set_drv_mode()	393
6.32.3.7	gpio_set_lvttl_mode()	393
6.32.3.8	gpio_set_value()	393
6.32.3.9	hsiom_set_config()	394
6.33	system/status.h File Reference	394
6.33.1	Detailed Description	395
6.33.2	Macro Definition Documentation	395
6.33.2.1	CCG_STATUS_CODE_OFFSET	395
6.33.3	Enumeration Type Documentation	395
6.33.3.1	ccg_status_t	395
6.34	system/system.h File Reference	396
6.34.1	Detailed Description	397
6.34.2	Enumeration Type Documentation	397
6.34.2.1	sys_fw_mode_t	397
6.34.3	Function Documentation	397
6.34.3.1	get_silicon_revision()	397
6.34.3.2	sys_get_bcdDevice_version()	398
6.34.3.3	sys_get_boot_version()	398
6.34.3.4	sys_get_custom_info_addr()	398
6.34.3.5	sys_get_device_mode()	398
6.34.3.6	sys_get_fw_img1_start_addr()	399
6.34.3.7	sys_get_fw_img2_start_addr()	399
6.34.3.8	sys_get_img1_fw_version()	399
6.34.3.9	sys_get_img2_fw_version()	400
6.34.3.10	sys_get_recent_fw_image()	400
6.34.3.11	sys_get_silicon_id()	400
6.34.3.12	sys_set_device_mode()	400
6.35	system/timer.h File Reference	401

6.35.1	Detailed Description	401
6.35.2	Typedef Documentation	402
6.35.2.1	timer_cb_t	402
6.35.2.2	timer_id_t	402
6.35.3	Function Documentation	402
6.35.3.1	timer_enter_sleep()	402
6.35.3.2	timer_get_count()	402
6.35.3.3	timer_get_multiplier()	403
6.35.3.4	timer_init()	403
6.35.3.5	timer_is_running()	403
6.35.3.6	timer_num_active()	404
6.35.3.7	timer_start()	404
6.35.3.8	timer_start_wocb()	404
6.35.3.9	timer_stop()	405
6.35.3.10	timer_stop_all()	405
6.35.3.11	timer_stop_range()	405
6.36	system/utils.h File Reference	406
6.36.1	Detailed Description	406
6.36.2	Macro Definition Documentation	407
6.36.2.1	MAKE_DWORD	407
6.36.2.2	REV_BYTE_ORDER	407
6.36.3	Function Documentation	407
6.36.3.1	crc16()	407
6.36.3.2	mem_calculate_byte_checksum()	408
6.36.3.3	mem_calculate_dword_checksum()	408
6.36.3.4	mem_calculate_word_checksum()	408
6.36.3.5	mem_copy_word()	409
6.37	usb/usb.h File Reference	409
6.37.1	Detailed Description	412
6.37.2	Macro Definition Documentation	412
6.37.2.1	USB_REMOTE_WAKEUP_TIMER_PERIOD	412
6.37.2.2	USB_SUSPEND_TIMER_PERIOD	412
6.37.2.3	USBDEV_ARB_EPx_CFG_CRC_BYPASS	412
6.37.2.4	USBDEV_ARB_EPx_CFG_DMA_REQ	412
6.37.2.5	USBDEV_ARB_EPx_CFG_IN_DATA_RDY	412
6.37.2.6	USBDEV_ARB_EPx_CFG_RESET_PTR	413
6.37.2.7	USBDEV_ARB_RWx_DR16_ADDRESS	413
6.37.2.8	USBDEV_ARB_RWx_DR_ADDRESS	413
6.37.2.9	USBDEV_SIE_EPx_CNT0_ADDRESS	413
6.37.2.10	USBDEV_SIE_EPx_CNT0_DATA_COUNT_MSB_MASK	413

6.37.2.11 USBDEV_SIE_EPx_CNT0_DATA_TOGGLE	413
6.37.2.12 USBDEV_SIE_EPx_CNT0_DATA_VALID	413
6.37.2.13 USBDEV_SIE_EPx_CNT1_ADDRESS	413
6.37.2.14 USBDEV_SIE_EPx_CNT1_DATA_COUNT_MASK	414
6.37.2.15 USBDEV_SIE_EPx_CR0_ACKED_TXN	414
6.37.2.16 USBDEV_SIE_EPx_CR0_ADDRESS	414
6.37.2.17 USBDEV_SIE_EPx_CR0_ERR_IN_TXN	414
6.37.2.18 USBDEV_SIE_EPx_CR0_MODE_MASK	414
6.37.2.19 USBDEV_SIE_EPx_CR0_NAK_INT_EN	414
6.37.2.20 USBDEV_SIE_EPx_CR0_STALL	414
6.37.3 Typedef Documentation	414
6.37.3.1 usb_event_cb_t	415
6.37.3.2 usb_setup_cb_t	415
6.37.4 Enumeration Type Documentation	415
6.37.4.1 usb_ep0_state_t	415
6.37.4.2 usb_ep_index_t	415
6.37.4.3 usb_state_t	416
6.37.4.4 usbdev_ep_mode_t	416
6.37.5 Function Documentation	416
6.37.5.1 usb_disable()	416
6.37.5.2 usb_enable()	417
6.37.5.3 usb_ep0_send_recv_status()	417
6.37.5.4 usb_ep0_set_stall()	417
6.37.5.5 usb_ep0_setup_read()	418
6.37.5.6 usb_ep0_setup_write()	418
6.37.5.7 usb_ep0_wait_for_ack()	419
6.37.5.8 usb_ep_clear_stall()	419
6.37.5.9 usb_ep_disable()	419
6.37.5.10 usb_ep_enable()	420
6.37.5.11 usb_ep_flush()	420
6.37.5.12 usb_ep_is_ready()	420
6.37.5.13 usb_ep_queue_read_single()	421
6.37.5.14 usb_ep_read_single()	421
6.37.5.15 usb_ep_send_zlp()	421
6.37.5.16 usb_ep_set_stall()	422
6.37.5.17 usb_ep_write_single()	422
6.37.5.18 usb_get_state()	423
6.37.5.19 usb_init()	423
6.37.5.20 usb_intr_lock()	423
6.37.5.21 usb_intr_unlock()	423

6.37.5.22 <code>usb_is_idle()</code>	424
6.37.5.23 <code>usb_remote_wakeup()</code>	424
6.37.5.24 <code>usb_task()</code>	424
6.37.5.25 <code>usb_vbus_int_handler()</code>	425
6.37.6 Variable Documentation	425
6.37.6.1 <code>USBARB</code>	425
6.37.6.2 <code>USBARB16</code>	425
6.37.6.3 <code>USBSIE</code>	425
6.38 <code>usb/usbconst.h</code> File Reference	425
6.38.1 Detailed Description	427
6.38.2 Macro Definition Documentation	427
6.38.2.1 <code>USB_TARGET_MASK</code>	427
6.38.3 Enumeration Type Documentation	427
6.38.3.1 <code>usb_dev_cap_type_t</code>	427
6.38.3.2 <code>usb_dscr_type_t</code>	427
6.38.3.3 <code>usb_ep_type_t</code>	428
6.38.3.4 <code>usb_feature_select_t</code>	428
6.38.3.5 <code>usb_setup_cmd_t</code>	428
Index	431

Chapter 1

CCGx Firmware Stack: API Reference Guide

1.1 Introduction

USB Type-C is the new USB-IF standard that solves several challenges faced when using today's Type-A and Type-B cables and connectors. USB Type-C uses a slimmer connector (measuring only 2.4-mm in height) to allow for increasing miniaturization of consumer and industrial products. The USB Type-C standard is gaining rapid support by enabling small form-factor, easy-to-use connectors and cables with the ability to transmit multiple protocols and offer power delivery up to 100 W – a significant improvement over the 7.5 W possible using previous standards.

1.1.1 USB Type-C Highlights

- Brand new reversible connector, measuring only 2.4-mm in height.
- Compliant with USB Power Delivery 2.0, providing up to 100 W.
- Double the bandwidth of USB 3.0, increasing to 10 Gbps with SuperSpeedPlus USB3.1.
- Combines multiple protocols in a single cable, including DisplayPort™, PCIe® or Thunderbolt™.

1.2 Cypress EZ-PD™ Type-C Controllers

Cypress provides the EZ-PD™ family of USB Type-C controllers which can help implement various Type-C applications. Cypress's Type-C controllers are based on Cypress's PSoC® 4 programmable system-on-chip architecture, which includes programmable analog and digital blocks, an ARM® Cortex®-M0 core and 32 to 128 KB of flash memory.

This product family is driving the industry's first Type-C products with top-tier PC makers, enabling them to bring these USB Type-C benefits to market. Cypress has reference designs readily available for EMCA and dongle applications. These are available online and could be used to speed-up our customers design cycle.

1.2.1 CCGx Product Families

The EZ-PD CCGx product line consists of the following product families:

- EZ-PD™ CCG1: Industry's First Programmable Type-C Port Controller
- EZ-PD™ CCG2: Industry's Smallest Programmable Type-C Port Controller
- EZ-PD™ CCG3: Industry's Most Integrated Type-C Port Controller
- EZ-PD™ CCG4: Industry's First Dual-Port Type-C Port Controller

- EZ-PD™ CCG5: Dual-Port Type-C and PD Port Controller for PCs and Docks
- EZ-PD™ CCG3PA: Power Delivery Solution for Adapters and Chargers

1.3 CCGx Firmware Stack

The CCGx product families are supported by a robust firmware stack which includes:

1. USB Type-C and USB-PD specification compliant PD stack.
2. Drivers for the various hardware blocks on the CCGx controllers.
3. Implementation of a Host Processor Interface (HPI) that allows an external embedded controller to monitor and control the CCGx device operation.
4. Implementation of the DisplayPort Alternate Mode that allows transfer of video signals over the Type-C data lanes.

The CCGx firmware stack is compliant to the:

- USB-PD Specification Revision 3.0
- Type-C Specification Revision 1.2

This version of the CCGx Firmware Stack supports the CCG3, CCG4 and CCG5 families and is targeted at implementing USB-PD port controllers for desktops and notebooks.

Please refer to the CCGx Power SDK for stack and code examples for implementing power adapters and chargers using the CCG3PA device family. Please use the CCGx SDK 3.0.2 version for stack and code examples for implementing Type-C video adapters using the CCG3 device family.

1.3.1 Firmware Stack Organization

The CCGx firmware stack is provided in source form with the SDK. The firmware files are organized into the following directory structure:

- system: The system folder contains header and source files relating to the CCGx device hardware and registers, boot-loader and flash access functions, low level drivers for the GPIO and USB-PD blocks on the CCGx device, and a soft timer implementation that is used by the firmware stack.
- pd_hal: The pd_hal folder contains the Hardware Adaptation Layer (HAL) or low level hardware driver for the USB-PD hardware in the CCGx device. The driver functionality include PD block initialization, USB-PD message handling code, interrupt handling and ADC/Comparator configuration.
- pd_common: The pd_common folder contains the core Type-C and USB-PD stack for the CCGx device. This includes the the Type-C port manager, the USB-PD protocol layer, the USB-PD policy engine and the Device Policy Manager. On devices that dual USB-PD ports, the stack allows both of the ports to function and be managed in a completely independent manner. The PD stack is provided in library form, and hence this folder will only contain the header files that define the stack interfaces.
- scb: The scb folder contains the driver code for I2C slave mode operation using the Serial Controller Blocks (SCB) on the CCGx device. Since I2C slave mode is the most commonly used interface for CCGx, a specially optimized driver is provided for the same. The SCB driver is provided in library form, and the folder only contains the header files that define the SCB driver interface.
- hpiss: The hpiss folder contains the implementation for the Host Processor Interface (HPI) which is an I2C based software protocol that allows an external Embedded Controller (EC) to monitor and control the CCGx device operation. HPI provides a register based interface through which the EC can manage the power contracts, send and receive VDMs and perform flash read/write operations. The HPI implementation is provided in library form and the folder only contains header files that define the interfaces. Please contact Cypress for more details about the HPI interface and its capabilities.

- app: The app folder contains the top-level application layer functionality that implements the USB-PD controller functions required. This includes functionality such as PDO evaluation and contract negotiation, VDM handling for both DFP and UFP roles, handling of control messages like role swap; and alternate mode discovery and negotiation. The alternate mode specific implementation can be found in the app/alt_mode directory.

Chapter 2

CCGx Firmware Architecture

The CCGx firmware solution allows users to implement a variety of USB-PD applications using the CCG devices and a fully tested firmware stack.

The CCGx firmware solution contains the following components:

- Hardware Abstraction Layer (HAL): This includes the low level drivers for the various hardware blocks on the CCG device. This includes drivers for the Type-C and USB-PD block, Serial Communication Blocks (SCBs), GPIOs, flash module and timer module.
- USB Type-C and USB-PD Protocol Stack: This is the complete USB-PD protocol stack that includes the Type-C and USB-PD port managers, USB-PD protocol layer, the USB-PD policy engine and the device policy manager. The device policy manager is designed to allow all policy decisions to be made at the application level, either on an external Embedded Controller (EC) or in the CCG firmware itself.
- Firmware update module: This is a firmware module that allows the device firmware maintained in internal flash to be updated. In Notebook PD port controller applications, the firmware update will be done from the EC side through an I2C interface.
- Host Processor Interface (HPI): The Host Processor Interface (HPI) is an I2C based control interface that allows an Embedded Controller (EC) to monitor and control the USB-PD port on the CCG device. The HPI is the means to allow the PC platform to control the PD policy management.
- Port Management: This module handles all of the PD port management functions including the algorithm for optimal contract negotiations, source and sink power control, source voltage selection, port role assignment and swap request handling.
- Alternate Modes: This module implements the alternate mode handling for CCG as a DFP and UFP. A fully tested implementation of DisplayPort alternate mode with CCG as DFP is provided. The module also allows users to implement their own alternate mode support in both DFP and UFP modes.
- Low Power: This module attempts to keep the CCG device in the low power standby mode as often as possible to minimize power consumption.
- External Hardware Control: This is an hardware design dependent module which controls the external hardware blocks such as FETs, regulators and Type-C switches.
- Solution specific tasks: This is an application layer module where any custom tasks required by the user solution can be implemented.

A block diagram of the CCGx firmware architecture is shown below.

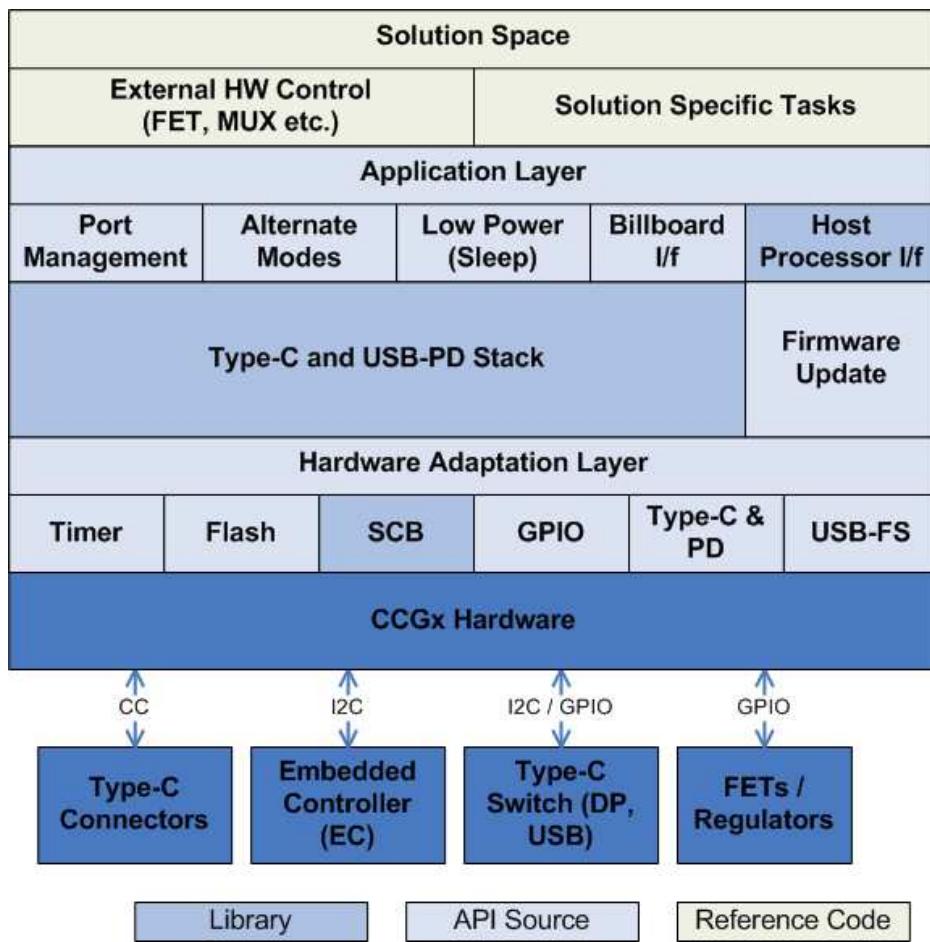


Figure 2.1: CCGx Firmware Stack Block Diagram

2.1 CCGx Firmware Solution Structure

The CCGx firmware solution is broken down into multiple layers:

- **Hardware Adaptation Layer:** The HAL is the lowest driver layer for the CCG device. The HAL code is provided in source form and can be located under the system folder in the CCGx firmware project.
- **USB-PD Stack:** The USB-PD stack includes the Type-C and PD managers, the PD protocol, policy engine and device policy manager blocks. The USB-PD stack is provided in the form of a pre-compiled library.
- **Application Layer:** The application layer includes the PD port management, HPI implementation, alternate modes and low power mode support. The application layer is provided in source form; and can be located under the app and alt_mode folders in the CCGx firmware project.
- **Solution Layer:** The solution layer includes external hardware management, the main task loop and any user application specific functionality. A reference implementation is provided under the solution folder in the CCGx firmware project; and is expected to be customized by users.

The CCGx firmware solution structure is shown below.

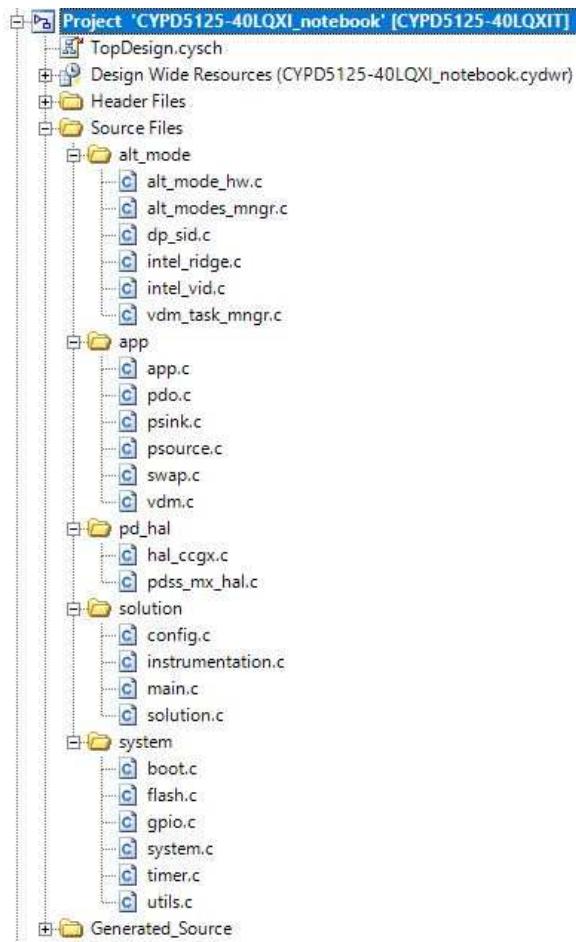


Figure 2.2: CCGx Firmware Solution Structure

2.2 Public API Summary

This document describes all of the data structures and functions that are part of the CCGx firmware stack. Many of these functions are intended to be used internally by the stack layers. The main public interfaces that are expected to be used by user code are summarized in this section.

2.2.1 Device Policy Manager (DPM) API

The Device Policy Manager (DPM) APIs are the public interfaces to the PD stack. These APIs include:

- [dpm_init\(\)](#)
- [dpm_start\(\)](#)
- [dpm_stop\(\)](#)
- [dpm_disable\(\)](#)
- [dpm_deepsleep\(\)](#)
- [dpm_wakeup\(\)](#)
- [dpm_task\(\)](#)
- [dpm_get_info\(\)](#)

- `dpm_update_def_cable_cap()`
- `dpm_get_def_cable_cap()`
- `dpm_update_snk_wait_cap_period()`
- `dpm_get_snk_wait_cap_period()`
- `dpm_update_mux_enable_wait_period()`
- `dpm_get_mux_enable_wait_period()`
- `dpm_pd_command()`
- `dpm_typec_command()`
- `dpm_update_swap_response()`
- `dpm_update_src_cap()`
- `dpm_update_src_cap_mask()`
- `dpm_update_snk_cap()`
- `dpm_update_snk_cap_mask()`
- `dpm_update_snk_max_min()`
- `dpm_update_port_config()`
- `dpm_get_polarity()`
- `dpm_typec_deassert_rp_rd()`
- `dpm_update_port_status()`
- `dpm_get_pd_port_status()`
- `dpm_update_frs_enable()`
- `dpm_update_ext_src_cap()`
- `dpm_prot_reset()`
- `dpm_send_hard_reset()`
- `dpm_get_sink_detach_margin()`
- `dpm_get_sink_detach_voltage()`

2.2.2 Host Processor Interface (HPI) API

The Host Processor Interface (HPI) API are functions that initialize the HPI register values and help perform message exchanges between CCG and the Embedded Controller (EC).

- `hpi_init()`
- `hpi_task()`
- `hpi_reg_enqueue_event()`
- `hpi_pd_event_handler()`
- `hpi_update_versions()`
- `hpi_set_mode_regs()`
- `hpi_update_fw_locations()`
- `hpi_sleep_allowed()`
- `hpi_sleep()`
- `hpi_get_port_enable()`

2.2.3 Application Layer API

The application layer is where the policy management for the CCGx application are implemented. The default implementations of these functions are based on the configuration table and the parameters provided by EC through the HPI registers.

- `app_init()`
- `app_task()`
- `app_event_handler()`
- `app_get_status()`
- `app_sleep()`
- `app_wakeup()`
- `system_sleep()`
- `eval_src_cap()`
- `eval_rdo()`
- `psnk_set_voltage()`
- `psnk_set_current()`
- `psnk_enable()`
- `psnk_disable()`
- `psrc_set_voltage()`
- `psrc_set_current()`
- `psrc_enable()`
- `psrc_disable()`
- `vconn_enable()`
- `vconn_disable()`
- `vconn_is_present()`
- `vbus_is_present()`
- `vbus_discharge_on()`
- `vbus_discharge_off()`
- `app_ovp_enable()`
- `app_ovp_disable()`
- `eval_dr_swap()`
- `eval_vconn_swap()`
- `eval_pr_swap()`
- `vdm_data_init()`
- `vdm_update_data()`
- `eval_vdm()`

2.2.3.1 Solution Layer Functions

The PD stack requires the user to provide a set of functions that provide the stack configuration and also handle operations that are hardware dependent. The following functions are expected to be implemented at the solution layer in user code.

- [mux_ctrl_init\(\)](#)
- [mux_ctrl_set_cfg\(\)](#)
- [sln_pd_event_handler\(\)](#)
- [app_get_callback_ptr\(\)](#)

2.2.4 Alternate Mode Manager API

The following APIs are provided by the CCG Alternate Mode Manager module. Some of these APIs are for use when CCG is a DFP, and the others are for use when CCG is an UFP.

- [enable_vdm_task_mngr\(\)](#)
- [vdm_task_mngr_deinit\(\)](#)
- [vdm_task_mngr\(\)](#)
- [is_vdm_task_idle\(\)](#)
- [eval_rec_vdm\(\)](#)

2.2.5 Hardware Abstraction Layer (HAL) API

This section documents the API provided as part of the Hardware Adaptation Layer (HAL) which provides drivers for various hardware blocks on the CCG device.

2.2.5.1 GPIO API

- [hsiom_set_config\(\)](#)
- [gpio_set_drv_mode\(\)](#)
- [gpio_hsiom_set_config\(\)](#)
- [gpio_int_set_config\(\)](#)
- [gpio_set_value\(\)](#)
- [gpio_read_value\(\)](#)
- [gpio_get_intr\(\)](#)
- [gpio_clear_intr\(\)](#)

2.2.5.2 I2C Driver API

- [i2c_scb_init\(\)](#)
- [i2c_scb_write\(\)](#)
- [i2c_reset\(\)](#)
- [i2c_slave_ack_ctrl\(\)](#)
- [i2c_scb_is_idle\(\)](#)
- [i2c_scb_enable_wakeup\(\)](#)

2.2.5.3 Flash Module API

- `flash_enter_mode()`
- `flash_access_get_status()`
- `flash_set_access_limits()`
- `flash_row_clear()`
- `flash_row_write()`
- `flash_row_read()`

2.2.5.4 Soft Timer API

- `timer_init()`
- `timer_start()`
- `timer_stop()`
- `timer_is_running()`
- `timer_stop_all()`
- `timer_stop_range()`
- `timer_num_active()`
- `timer_enter_sleep()`

2.2.6 Firmware Update API

These APIs provide the basic boot and firmware upgrade related functions that are used by various firmware update protocol implementations like HPI and CC unstructured command handlers.

- `boot_validate_fw()`
- `boot_validate_configtable()`
- `get_boot_mode_reason()`
- `boot_get_boot_seq()`
- `sys_set_device_mode()`
- `sys_get_device_mode()`

Please see the detailed description of the APIs in the following chapters. You can also refer to the CCGx SDK User Guide document that provides some usage examples and guidelines.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

pd_do_t::ACT_CBL_VDO	19
alt_mode_evt_t::ALT_MODE_EVT	21
Struct containing alternate modes manager event/command	
alt_mode_evt_t::ALT_MODE_EVT_DATA	22
Struct containing alternate modes manager event/command data	
alt_mode_evt_t	22
Alt modes manager application event/command structure	
alt_mode_hw_evt_t::ALT_MODE_HW_EVT	23
Struct containing alternate modes HW event/command	
alt_mode_hw_evt_t	24
Alt mode HW application event/command structure	
alt_mode_info_t	25
This structure holds all necessary information for interaction between alt modes manager and selected alternative mode	
alt_mode_reg_info_t	27
This structure holds all necessary information on Discovery Mode stage for supported alternative mode when alt modes manager registers new alt mode	
app_cbk_t	29
Struct to hold the application interface. The application is expected to fill the structure with pointers to functions that use the on-board circuitry to accomplish tasks like source/sink power turn on/off. All the functions in this structure must be non-blocking and take minimum execution time	
app_resp_t	32
Struct to hold response to policy manager. Note: The app_resp_t structure is only used for responses that have a single DO. This may need to get extended if more DOs are to be supported	
app_status_t	33
This structure hold all variables related to application layer functionality	
atch_tgt_info_t	37
Struct holds Discovery info for alt modes manager	
bat_chg_params_t	38
Struct to hold the battery charging parameters	
pd_do_t::BAT_SNK	39
pd_do_t::BAT_SRC	40
bb_handle_t	40
Internal billboard module handle structure	
bb_settings_t	43
Struct to hold the Billboard settings	
pd_do_t::BIST_DO	45

<code>cc_state_t</code>	Union to hold CC status	45
<code>cg_timer_t</code>	Structure encapsulating information relating to a software timer	46
<code>chg_cfg_params_t</code>	Struct to hold the port legacy charging parameters	47
<code>comp_tbl_t</code>	This structure are used in alt_modes_config.h file to set alternative modes compatibility and priority	49
<code>contract_t</code>	Structure to hold PD Contract information	49
<code>pd_do_t::DP_CONFIG_VDO</code>	50
<code>pd_do_t::DP_STATUS_VDO</code>	51
<code>dpm_pd_cmd_buf_t</code>	Struct to hold PD command buffer	52
<code>dpm_status_t</code>	PD Device Policy Status structure. This structure holds all of the configuration and status information associated with a port on the CCG device. Members of this structure should not be directly modified by any of the application code	54
<code>pd_do_t::FIXED_SNK</code>	68
<code>pd_do_t::FIXED_SRC</code>	70
<code>fw_img_status_t</code>	Boot mode reason structure	72
<code>fw_img_status_t::fw_mode_reason_t</code>	73
<code>i2c_scb_config_t</code>	This structure holds all configuration associated with each I2C block	73
<code>ocp_settings_t</code>	Struct to hold the OCP settings	75
<code>otp_settings_t</code>	Struct to hold the OTP settings	76
<code>ovp_settings_t</code>	Struct to hold the OVP settings	78
<code>pd_do_t::PAS_CBL_VDO</code>	79
<code>pd_config_t</code>	Struct to hold the PD device configuration	80
<code>pd_contract_info_t</code>	Stucture to hold PD Contract information passed with APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE event to the application	83
<code>pd_do_t</code>	Union to hold a PD data object. All USB-PD data objects are 4-byte values which are interpreted according to the message type, length and object position. This union represents all possible interpretations of a USB-PD data object	84
<code>pd_extd_hdr_t</code>	Union to hold the PD extended header	89
<code>pd_hdr_t::PD_HDR</code>	90
<code>pd_hdr_t</code>	Union to hold the PD header defined by the USB-PD specification. Lower 16 bits hold the message header and the upper 16 bits hold the extended message header (where applicable)	92
<code>pd_packet_extd_t</code>	Struct to hold extended PD packets (messages)	92
<code>pd_packet_t</code>	Struct to hold a PD packet	93
<code>pd_port_config_t</code>	PD port-specific configuration data from the configuration table	95
<code>pd_port_status_t::PD_PORT_STAT</code>	103
<code>pd_port_status_t</code>	PD port status as reported to Embedded Controller	106

pd_power_status_t	PD port status corresponding to the Status Data Block (SSDB) See Table 6-39 of USB-PD R3 specification	107
pwr_params_t	Struct to hold the port power parameters	108
pd_do_t::QC_4_0_DATA_VDO		109
pd_do_t::RDO_BAT		110
pd_do_t::RDO_BAT_GIVEBACK		111
pd_do_t::RDO_FIXED_VAR		113
pd_do_t::RDO_FIXED_VAR_GIVEBACK		114
pd_do_t::RDO_GEN		116
pd_do_t::RDO_GEN_GVB		118
ridge_reg_t	Over-Current status bit in the status register	119
scp_settings_t	Struct to hold the SCP settings	120
pd_do_t::SRC_GEN		121
pd_do_t::STD_AMA_VDO		122
pd_do_t::STD_AMA_VDO_PD3		123
pd_do_t::STD_CBL_VDO		124
pd_do_t::STD_CERT_VDO		126
pd_do_t::STD_DP_VDO		127
pd_do_t::STD_PROD_VDO		128
pd_do_t::STD_SVID_RESP_VDO		129
pd_do_t::STD_VDM_HDR		129
pd_do_t::STD_VDM_ID_HDR		130
sys_fw_metadata_t	CCGx Firmware metadata structure	131
usb_config_t	USB device mode configuration information	133
usb_ep_handle_t	USB endpoint handle	135
usb_handle_t	USB device controller handle	135
usb_i2cm_t	Internal USB-I2CM module handle structure	138
usb_setup_pkt_t	USB setup packet format	139
USBARB16_REGS_T		140
USBARB_REGS_T		141
ridge_reg_t::USBPD_CMD_REG	Struct containing USB-PD controller command register fields for Alpine/Titan Ridge	141
ridge_reg_t::USBPD_STATUS_REG	Struct containing USB-PD controller status to be reported to Alpine/Titan Ridge. Using the structure definition corresponding to Titan Ridge in all cases	143
USBSIE_REGS_T		147
pd_do_t::USTD_QC_4_0_HDR		148
pd_do_t::USTD_VDM_HDR		148
uvp_settings_t	Struct to hold the UVP settings	150
pd_do_t::VAR_SNK		150
pd_do_t::VAR_SRC		151
vconn_ocp_settings_t	Struct to hold the Vconn OCP settings	152
vdm_msg_info_t	Struct holds received/sent VDM info	153
vdm_resp_t	Struct to hold response to policy manager	154

Chapter 4

File Index

4.1 File List

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

app/app.h	PD application handler header file	181
app/billboard.h	Billboard control interface header file	196
app/pdo.h	PDO evaluation and handler definitions	204
app/psink.h	Power Sink (Consumer) manager header file	205
app/psource.h	Power source (Provider) manager header file	207
app/swap.h	Swap request (PR_SWAP, DR_SWAP, VCONN_SWAP) handlers	213
app/usb_hid.h	USB HID class interface (flashing) header file	214
app/usb_i2cm.h	USB I2C master bridge interface header file	220
app/uvdm.h	Unstructured VDM handler header file	223
app/vdm.h	Vendor Defined Message (VDM) handler header file	226
app/alt_mode/alt_mode_hw.h	Hardware control for Alternate mode implementation	155
app/alt_mode/alt_modes_mngr.h	Alternate Mode Manager header file	161
app/alt_mode/dp_sid.h	DisplayPort alternate mode handler header file	170
app/alt_mode/intel_ridge.h	Intel Alpine/Titan Ridge control interface header file	173
app/alt_mode/intel_vid.h	Thunderbolt (Intel VID) alternate mode handler header file	176
app/alt_mode/vdm_task_mngr.h	VDM task manager header file	178
app/ridge_slave/ridge_slave.h	Alpine/Titan Ridge I2C slave interface header file	209
hpiss/hpi.h	Host Processor Interface (HPI) header file	228
pd_common/dpm.h	Device Policy Manager (DPM) header file	243

pd_common/pd.h	Common definitions and structures used in the CCG USB-PD stack	260
pd_common/pd_policy_engine.h	USB-PD policy engine header file	307
pd_common/pd_protocol.h	USB-PD protocol layer header file	311
pd_common/typec_manager.h	Type-C manager header file	318
pd_hal/hal_ccgx.h	PD and Type-C HAL layer for CCGx device family	322
pd_hal/hpd.h	Hotplug Detect (HPD) driver header file	326
pd_hal/pdss_hal.h	CCG PD PHY driver module header file	330
scb/i2c.h	I2C slave driver header file	370
system/boot.h	Bootloader support header file	376
system/ccgx_api_desc.h	CCGx Firmware API Reference Guide Introduction	380
system/ccgx_version.h	This file defines the base firmware stack version for CCGx	380
system/flash.h	Flash command handler header file	381
system/gpio.h	GPIO and IO mapping control functions	386
system/status.h	API return status definitions	394
system/system.h	Support functions and definitions for bootloader and flash updates	396
system/timer.h	Soft timer header file	401
system/utils.h	General utility macros and definitions for CCGx firmware stack	406
usb/usb.h	USB driver header file for CCG3 & DMC (Dock Management Controller)	409
usb/usbconst.h	Constants defined in the USB specification	425

Chapter 5

Data Structure Documentation

5.1 pd_do_t::ACT_CBL_VDO Struct Reference

Data Fields

- uint32_t `usb_ss_sup`: 3
- uint32_t `sop_dp`: 1
- uint32_t `vbus_thru_cbl`: 1
- uint32_t `vbus_cur`: 2
- uint32_t `rsvd1`: 2
- uint32_t `max_vbus_volt`: 2
- uint32_t `cbl_term`: 2
- uint32_t `cbl_latency`: 4
- uint32_t `typec_plug`: 1
- uint32_t `typec_abc`: 2
- uint32_t `rsvd2`: 1
- uint32_t `vdo_version`: 3
- uint32_t `cbl_fw_ver`: 4
- uint32_t `cbl_hw_ver`: 4

5.1.1 Field Documentation

5.1.1.1 `cbl_fw_ver`

`uint32_t cbl_fw_ver`

Cable firmware version.

5.1.1.2 `cbl_hw_ver`

`uint32_t cbl_hw_ver`

Cable hardware version.

5.1.1.3 `cbl_latency`

`uint32_t cbl_latency`

Cable latency.

5.1.1.4 **cbl_term**

uint32_t cbl_term

Cable termination and VConn power requirement.

5.1.1.5 **max_vbus_volt**

uint32_t max_vbus_volt

Max. VBus voltage supported.

5.1.1.6 **rsvd1**

uint32_t rsvd1

Reserved field.

5.1.1.7 **rsvd2**

uint32_t rsvd2

Reserved field.

5.1.1.8 **sop_dp**

uint32_t sop_dp

Whether SOP" controller is present.

5.1.1.9 **typec_abc**

uint32_t typec_abc

Cable plug type.

5.1.1.10 **typec_plug**

uint32_t typec_plug

Reserved field.

5.1.1.11 **usb_ss_sup**

uint32_t usb_ss_sup

USB signalling supported by the cable.

5.1.1.12 **vbus_cur**

uint32_t vbus_cur

VBus current supported by the cable.

5.1.1.13 vbus_thru_cbl

```
uint32_t vbus_thru_cbl
```

Whether cable conducts VBus through.

5.1.1.14 vdo_version

```
uint32_t vdo_version
```

VDO version.

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

- pd_common/[pd.h](#)

5.2 alt_mode_evt_t::ALT_MODE_EVT Struct Reference

```
#include <alt_modes_mngr.h>
```

Data Fields

- uint32_t [data_role](#): 1
- uint32_t [alt_mode_evt](#): 7
- uint32_t [alt_mode](#): 8
- uint32_t [svid](#): 16

5.2.1 Detailed Description

Struct containing alternate modes manager event/command .

5.2.2 Field Documentation

5.2.2.1 alt_mode

```
uint32_t alt_mode
```

Alt mode ID.

5.2.2.2 alt_mode_evt

```
uint32_t alt_mode_evt
```

Alt mode event index from alt_mode_app_evt_t structure.

5.2.2.3 data_role

```
uint32_t data_role
```

Current event sender data role.

5.2.2.4 svid

```
uint32_t svid
```

Alt mode related SVID.

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.3 alt_mode_evt_t::ALT_MODE_EVT_DATA Struct Reference

```
#include <alt_modes_mngr.h>
```

Data Fields

- uint32_t [evt_data](#): 24
- uint32_t [evt_type](#): 8

5.3.1 Detailed Description

Struct containing alternate modes manager event/command data.

5.3.2 Field Documentation

5.3.2.1 evt_data

```
uint32_t evt_data
```

Alt mode event's data.

5.3.2.2 evt_type

```
uint32_t evt_type
```

Alt mode event's type.

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.4 alt_mode_evt_t Union Reference

```
#include <alt_modes_mngr.h>
```

Data Structures

- struct [ALT_MODE_EVT](#)
- struct [ALT_MODE_EVT_DATA](#)

Data Fields

- uint32_t [val](#)
- struct [alt_mode_evt_t::ALT_MODE_EVT alt_mode_event](#)
- struct [alt_mode_evt_t::ALT_MODE_EVT_DATA alt_mode_event_data](#)

5.4.1 Detailed Description

Alt modes manager application event/command structure.

5.4.2 Field Documentation

5.4.2.1 alt_mode_event

```
struct alt_mode_evt_t::ALT_MODE_EVT alt_mode_event
```

Union containing the alt mode event value.

5.4.2.2 alt_mode_event_data

```
struct alt_mode_evt_t::ALT_MODE_EVT_DATA alt_mode_event_data
```

Union containing the alt mode event's data value.

5.4.2.3 val

```
uint32_t val
```

Integer field used for direct manipulation of reason code.

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.5 alt_mode_hw_evt_t::ALT_MODE_HW_EVT Struct Reference

```
#include <alt_mode_hw.h>
```

Data Fields

- uint32_t [evt_data](#): 16
- uint32_t [hw_type](#): 8
- uint32_t [data_role](#): 8

5.5.1 Detailed Description

Struct containing alternate modes HW event/command .

5.5.2 Field Documentation

5.5.2.1 data_role

```
uint32_t data_role
```

Current data role.

5.5.2.2 evt_data

```
uint32_t evt_data
```

Current event/command data.

5.5.2.3 hw_type

```
uint32_t hw_type
```

HW type event/command related to.

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

- app/alt_mode/[alt_mode_hw.h](#)

5.6 alt_mode_hw_evt_t Union Reference

```
#include <alt_mode_hw.h>
```

Data Structures

- struct [ALT_MODE_HW_EVT](#)

Data Fields

- uint32_t [val](#)
- struct [alt_mode_hw_evt_t::ALT_MODE_HW_EVT](#) [hw_evt](#)

5.6.1 Detailed Description

Alt mode HW application event/command structure.

5.6.2 Field Documentation

5.6.2.1 hw_evt

```
struct alt\_mode\_hw\_evt\_t::ALT\_MODE\_HW\_EVT hw_evt
```

Union containing the application HW event/command value.

5.6.2.2 val

```
uint32_t val
```

Integer field used for direct manipulation of reason code.

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

- app/alt_mode/alt_mode_hw.h

5.7 alt_mode_info_t Struct Reference

```
#include <alt_modes_mngr.h>
```

Data Fields

- alt_mode_state_t mode_state
- alt_mode_state_t sop_state [SOP_DPRIME+1]
- uint8_t vdo_max_numb
- uint8_t obj_pos
- uint8_t cbl_obj_pos
- uint8_t alt_mode_id
- pd_do_t vdm_header
- pd_do_t * vdo [SOP_DPRIME+1]
- uint8_t vdo_numb [SOP_DPRIME+1]
- alt_mode_cbk_t cbk
- bool is_active
- bool custom_att_obj_pos
- bool uvd़m_supp
- bool set_mux_isolate
- bool app_evt_needed
- alt_mode_evt_t app_evt_data
- alt_mode_app_cbk_t eval_app_cmd

5.7.1 Detailed Description

This structure holds all necessary information for interaction between alt modes manager and selected alternative mode.

5.7.2 Field Documentation

5.7.2.1 alt_mode_id

```
uint8_t alt_mode_id
```

Alternative mode ID.

5.7.2.2 app_evt_data

```
alt_mode_evt_t app_evt_data
```

APP event data.

5.7.2.3 app_evt_needed

`bool app_evt_needed`

APP event flag.

5.7.2.4 cbk

`alt_mode_cbk_t cbk`

Alternate mode callback function.

5.7.2.5 cbl_obj_pos

`uint8_t cbl_obj_pos`

Cabel object position.

5.7.2.6 custom_att_obj_pos

`bool custom_att_obj_pos`

Object position field in Att VDM used by alt mode as custom.

5.7.2.7 eval_app_cmd

`alt_mode_app_cbk_t eval_app_cmd`

APP command cbk.

5.7.2.8 is_active

`bool is_active`

Active mode flag.

5.7.2.9 mode_state

`alt_mode_state_t mode_state`

Alternate mode state.

5.7.2.10 obj_pos

`uint8_t obj_pos`

Alternate mode object position.

5.7.2.11 set_mux_isolate

`bool set_mux_isolate`

Flag to indicate if MUX should be set to isolate while ENTER/EXIT alt mode

5.7.2.12 sop_state

```
alt_mode_state_t sop_state[SOP_DPRIME+1]
```

VDM state for SOP/SOP'/SOP" packets.

5.7.2.13 uvdm_supp

```
bool uvdm_supp
```

Flag to indicate if alt mode support unstructured VDMs.

5.7.2.14 vdm_header

```
pd_do_t vdm_header
```

Buffer to hold VDM header.

5.7.2.15 vdo

```
pd_do_t* vdo[SOP_DPRIME+1]
```

Pointers array to alt mode VDO buffers

5.7.2.16 vdo_max_numb

```
uint8_t vdo_max_numb
```

Maximum number of VDO that alt mode can handle

5.7.2.17 vdo_numb

```
uint8_t vdo_numb[SOP_DPRIME+1]
```

Current number of VDOs used for processing in VDO buffers

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.8 alt_mode_reg_info_t Struct Reference

```
#include <alt_modes_mngr.h>
```

Data Fields

- uint8_t atch_type
- uint8_t data_role
- uint8_t alt_mode_id
- [sop_t](#) cbl_sop_flag
- [pd_do_t](#) svid_emca_vdo
- [pd_do_t](#) svid_vdo
- const [atch_tgt_info_t](#) * atch_tgt_info
- [alt_mode_app_evt_t](#) app_evt

5.8.1 Detailed Description

This structure holds all necessary information on Discovery Mode stage for supported alternative mode when alt modes manager registers new alt mode.

5.8.2 Field Documentation

5.8.2.1 alt_mode_id

```
uint8_t alt_mode_id
```

Alt mode ID.

5.8.2.2 app_evt

```
alt_mode_app_evt_t app_evt
```

APP event.

5.8.2.3 atch_tgt_info

```
const atch_tgt_info_t* atch_tgt_info
```

Attached targets info (dev/ama/cbl) from Discovery ID response.

5.8.2.4 atch_type

```
uint8_t atch_type
```

Target of disc svid (cable or device/ama).

5.8.2.5 cbl_sop_flag

```
sop_t cbl_sop_flag
```

Sop indication flag.

5.8.2.6 data_role

```
uint8_t data_role
```

Current data role.

5.8.2.7 svid_emca_vdo

```
pd_do_t svid_emca_vdo
```

SVID VDO from cable Discovery mode response.

5.8.2.8 svid_vdo

```
pd_do_t svid_vdo
```

SVID VDO from Discovery mode SOP response.

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.9 app_cbk_t Struct Reference

```
#include <pd.h>
```

Data Fields

- void(* [app_event_handler](#))(uint8_t port, [app_evt_t](#) evt, const void *dat)
- void(* [psrc_set_voltage](#))(uint8_t port, uint16_t volt_mV)
- void(* [psrc_set_current](#))(uint8_t port, uint16_t cur_10mA)
- void(* [psrc_enable](#))(uint8_t port, [pwr_ready_cbk_t](#) pwr_ready_handler)
- void(* [psrc_disable](#))(uint8_t port, [pwr_ready_cbk_t](#) pwr_ready_handler)
- void(* [vconn_enable](#))(uint8_t port, uint8_t channel)
- void(* [vconn_disable](#))(uint8_t port, uint8_t channel)
- bool(* [vconn_is_present](#))(uint8_t port)
- bool(* [vbus_is_present](#))(uint8_t port, uint16_t volt, int8_t per)
- void(* [vbus_discharge_on](#))(uint8_t port)
- void(* [vbus_discharge_off](#))(uint8_t port)
- void(* [psnk_set_voltage](#))(uint8_t port, uint16_t volt_mV)
- void(* [psnk_set_current](#))(uint8_t port, uint16_t cur_10mA)
- void(* [psnk_enable](#))(uint8_t port)
- void(* [psnk_disable](#))(uint8_t port, [sink_discharge_off_cbk_t](#) snk_discharge_off_handler)
- void(* [eval_src_cap](#))(uint8_t port, const [pd_packet_t](#) *src_cap, [app_resp_cbk_t](#) app_resp_handler)
- void(* [eval_rdo](#))(uint8_t port, [pd_do_t](#) rdo, [app_resp_cbk_t](#) app_resp_handler)
- void(* [eval_dr_swap](#))(uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- void(* [eval_pr_swap](#))(uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- void(* [eval_vconn_swap](#))(uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- void(* [eval_vdm](#))(uint8_t port, const [pd_packet_t](#) *vdm, [vdm_resp_cbk_t](#) vdm_resp_handler)
- void(* [eval_fr_swap](#))(uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- uint16_t(* [vbus_get_value](#))(uint8_t port)

5.9.1 Detailed Description

Struct to hold the application interface. The application is expected to fill the structure with pointers to functions that use the on-board circuitry to accomplish tasks like source/sink power turn on/off. All the functions in this structure must be non-blocking and take minimum execution time.

Warning

The application must check the callback pointer passed by the stack is not NULL.

5.9.2 Field Documentation

5.9.2.1 app_event_handler

```
void(* app_event_handler) (uint8_t port, app_evt_t evt, const void *dat)
```

App event handler callback.

5.9.2.2 eval_dr_swap

```
void(* eval_dr_swap) (uint8_t port, app_resp_cbk_t app_resp_handler)
```

Handles DR swap request received by port.

5.9.2.3 eval_fr_swap

```
void(* eval_fr_swap) (uint8_t port, app_resp_cbk_t app_resp_handler)
```

Handle FR swap request received by the specified port.

5.9.2.4 eval_pr_swap

```
void(* eval_pr_swap) (uint8_t port, app_resp_cbk_t app_resp_handler)
```

Handles pr swap request received by port.

5.9.2.5 eval_rdo

```
void(* eval_rdo) (uint8_t port, pd_do_t rdo, app_resp_cbk_t app_resp_handler)
```

Evaluate sink request message.

5.9.2.6 eval_src_cap

```
void(* eval_src_cap) (uint8_t port, const pd_packet_t *src_cap, app_resp_cbk_t app_resp_handler)
```

Evaluate received source caps and provide the RDO to be used to negotiate contract.

5.9.2.7 eval_vconn_swap

```
void(* eval_vconn_swap) (uint8_t port, app_resp_cbk_t app_resp_handler)
```

Handles VCONN swap request received by port.

5.9.2.8 eval_vdm

```
void(* eval_vdm) (uint8_t port, const pd_packet_t *vdm, vdm_resp_cbk_t vdm_resp_handler)
```

Handle VDMs (all structured/unstructured VDMs need to be handled) received by the port.

5.9.2.9 psnk_disable

```
void(* psnk_disable) (uint8_t port, sink_discharge_off_cbk_t snk_discharge_off_handler)
```

Disable power sink related circuitry.

5.9.2.10 psnk_enable

```
void(* psnk_enable) (uint8_t port)
```

Enable power sink related circuitry.

5.9.2.11 psnk_set_current

```
void(* psnk_set_current) (uint8_t port, uint16_t cur_10mA)
```

Set power sink current, in 10mA units.

5.9.2.12 psnk_set_voltage

```
void(* psnk_set_voltage) (uint8_t port, uint16_t volt_mV)
```

Set power sink voltage, in mV units.

5.9.2.13 psrc_disable

```
void(* psrc_disable) (uint8_t port, pwr_ready_cbk_t pwr_ready_handler)
```

Disable the power supply. The pwr_ready_handler, if not NULL, must be called when the power supply has been discharged to Vsafe0V.

5.9.2.14 psrc_enable

```
void(* psrc_enable) (uint8_t port, pwr_ready_cbk_t pwr_ready_handler)
```

Enable the power supply. The pwr_ready_handler, if not NULL, must be called when the power supply is ready.

5.9.2.15 psrc_set_current

```
void(* psrc_set_current) (uint8_t port, uint16_t cur_10mA)
```

Set power source current in 10mA units.

5.9.2.16 psrc_set_voltage

```
void(* psrc_set_voltage) (uint8_t port, uint16_t volt_mV)
```

Set power source voltage in mV units.

5.9.2.17 vbus_discharge_off

```
void(* vbus_discharge_off) (uint8_t port)
```

Turn off VBUS discharge circuit.

5.9.2.18 vbus_discharge_on

```
void(* vbus_discharge_on) (uint8_t port)
```

Turn on VBUS discharge circuit.

5.9.2.19 vbus_get_value

```
uint16_t(* vbus_get_value) (uint8_t port)
```

Get current VBUS value in mV from application.

5.9.2.20 vbus_is_present

```
bool(* vbus_is_present) (uint8_t port, uint16_t volt, int8_t per)
```

Check whether VBus voltage is within expected range.

5.9.2.21 vconn_disable

```
void(* vconn_disable) (uint8_t port, uint8_t channel)
```

Turn VCONN supply OFF.

5.9.2.22 vconn_enable

```
void(* vconn_enable) (uint8_t port, uint8_t channel)
```

Turn VCONN supply ON.

5.9.2.23 vconn_is_present

```
bool(* vconn_is_present) (uint8_t port)
```

Check whether VConn supply is ON.

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

- pd_common/[pd.h](#)

5.10 app_resp_t Struct Reference

```
#include <pd.h>
```

Data Fields

- [pd_do_t resp_do](#)
- [app_req_status_t req_status](#)

5.10.1 Detailed Description

Struct to hold response to policy manager. Note: The [app_resp_t](#) structure is only used for responses that have a single DO. This may need to get extended if more DOs are to be supported.

5.10.2 Field Documentation

5.10.2.1 req_status

```
app_req_status_t req_status
```

Request status.

5.10.2.2 resp_do

```
pd_do_t resp_do
```

Response data object.

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

- pd_common/pd.h

5.11 app_status_t Struct Reference

```
#include <app.h>
```

Data Fields

- pwr_ready_cbk_t pwr_ready_cbk
- sink_discharge_off_cbk_t snk_dis_cbk
- app_resp_t app_resp
- vdm_resp_t vdm_resp
- uint16_t psrc_volt
- uint16_t psrc_volt_old
- uint16_t psnk_volt
- uint16_t psnk_cur
- uint8_t vdm_task_en
- uint8_t cbl_disc_id_finished
- uint8_t vdm_version
- uint8_t alt_mode_trig_mask
- volatile uint8_t fault_status
- bool alt_mode_entered
- bool vdm_prcs_failed
- bool is_vbus_on
- bool is_vconn_on
- bool vdm_retry_pending
- bool psrc_rising
- bool cur_fb_enabled
- bool ld_sw_ctrl
- bool bc_12_src_disabled
- bool is_mux_busy
- vdm_resp_cbk_t vdm_resp_cbk
- bool is_vdm_pending
- mux_poll_fnc_cbk_t mux_poll_cbk

5.11.1 Detailed Description

This structure hold all variables related to application layer functionality.

5.11.2 Field Documentation

5.11.2.1 alt_mode_entered

```
bool alt_mode_entered
```

Flag to indicate is alternate modes currently entered.

5.11.2.2 alt_mode_trig_mask

```
uint8_t alt_mode_trig_mask
```

Mask to indicate which alt mode should be enabled by EC.

5.11.2.3 app_resp

```
app_resp_t app_resp
```

Buffer for APP responses.

5.11.2.4 bc_12_src_disabled

```
bool bc_12_src_disabled
```

BC 1.2 source disabled flag.

5.11.2.5 cbl_disc_id_finished

```
uint8_t cbl_disc_id_finished
```

Flag to indicate that cable disc id finished.

5.11.2.6 cur_fb_enabled

```
bool cur_fb_enabled
```

Indicates that current foldback is enabled

5.11.2.7 fault_status

```
volatile uint8_t fault_status
```

Fault status bits for this port.

5.11.2.8 is_mux_busy

```
bool is_mux_busy
```

Flag to indicate that mux is switching.

5.11.2.9 is_vbus_on

```
bool is_vbus_on
```

Is supplying VBUS flag.

5.11.2.10 is_vconn_on

```
bool is_vconn_on
```

Is supplying VCONN flag.

5.11.2.11 is_vdm_pending

```
bool is_vdm_pending
```

VDM handling flag for MUX callback.

5.11.2.12 ld_sw_ctrl

```
bool ld_sw_ctrl
```

Indicates whether the VBUS load switch control is active or not.

5.11.2.13 mux_poll_cbk

```
mux\_poll\_fnc\_cbk\_t mux_poll_cbk
```

Holds pointer to MUX polling function.

5.11.2.14 psnk_cur

```
uint16_t psnk_cur
```

Current PSink current in 10mA units.

5.11.2.15 psnk_volt

```
uint16_t psnk_volt
```

Current PSink voltage in mV units.

5.11.2.16 psrc_rising

```
bool psrc_rising
```

Voltage ramp up/down.

5.11.2.17 psrc_volt

```
uint16_t psrc_volt
```

Current Psource voltage in mV

5.11.2.18 psrc_volt_old

```
uint16_t psrc_volt_old
```

Old Psource voltage in mV

5.11.2.19 pwr_ready_cbk

```
pwr_ready_cbk_t pwr_ready_cbk
```

Registered Power source callback.

5.11.2.20 snk_dis_cbk

```
sink_discharge_off_cbk_t snk_dis_cbk
```

Registered Power sink callback.

5.11.2.21 vdm_prcs_failed

```
bool vdm_prcs_failed
```

Flag to indicate is vdm process failed.

5.11.2.22 vdm_resp

```
vdm_resp_t vdm_resp
```

Buffer for VDM responses.

5.11.2.23 vdm_resp_cbk

```
vdm_resp_cbk_t vdm_resp_cbk
```

VDM response handler callback.

5.11.2.24 vdm_retry_pending

```
bool vdm_retry_pending
```

Whether VDM retry on timeout is pending.

5.11.2.25 vdm_task_en

```
uint8_t vdm_task_en
```

Flag to indicate is vdm task manager enabled.

5.11.2.26 vdm_version

```
uint8_t vdm_version
```

Live VDM version.

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

- [app/app.h](#)

5.12 atch_tgt_info_t Struct Reference

```
#include <vdm_task_mngr.h>
```

Data Fields

- `pd_do_t tgt_id_header`
- `pd_do_t ama_vdo`
- `const pd_do_t * cbl_vdo`
- `uint16_t tgt_svrid[MAX_SVID_VDO_SUPP]`
- `uint16_t cbl_svrid[MAX_SVID_VDO_SUPP]`

5.12.1 Detailed Description

Struct holds Discovery info for alt modes manager.

This struct holds Discovery information which uses by alt modes manager.

5.12.2 Field Documentation

5.12.2.1 ama_vdo

```
pd_do_t ama_vdo
```

Holds AMA discovery ID response VDO .

5.12.2.2 cbl_svrid

```
uint16_t cbl_svrid[MAX_SVID_VDO_SUPP]
```

Holds received SVID for cable .

5.12.2.3 cbl_vdo

```
const pd_do_t* cbl_vdo
```

Pointer to cable VDO .

5.12.2.4 tgt_id_header

```
pd_do_t tgt_id_header
```

Holds dev/ama discovery ID header .

5.12.2.5 tgt_svrid

```
uint16_t tgt_svrid[MAX_SVID_VDO_SUPP]
```

Holds received SVID for dev/ama .

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

- app/alt_mode/vdm_task_mngr.h

5.13 bat_chg_params_t Struct Reference

```
#include <pd.h>
```

Data Fields

- `uint8_t table_len`
- `uint8_t reserved_0`
- `uint16_t vbatt_max_volt`
- `uint16_t vbatt_cutoff_volt`
- `uint16_t vbatt_dischg_en_volt`
- `uint16_t vbatt_max_cur`
- `uint16_t reserved_1`

5.13.1 Detailed Description

Struct to hold the battery charging parameters.

5.13.2 Field Documentation

5.13.2.1 reserved_0

```
uint8_t reserved_0
```

Reserved for future use

5.13.2.2 reserved_1

```
uint16_t reserved_1
```

Reserved for future use

5.13.2.3 table_len

```
uint8_t table_len
```

Battery charger parameter table length in bytes

5.13.2.4 vbatt_cutoff_volt

```
uint16_t vbatt_cutoff_volt
```

Minimum battery voltage below which device should stop discharging.

5.13.2.5 vbatt_dischg_en_volt

```
uint16_t vbatt_dischg_en_volt
```

Battery voltage at which discharge can be re-enabled.

5.13.2.6 vbatt_max_cur

```
uint16_t vbatt_max_cur
```

Maximum charging current allowed for the battery in mA.

5.13.2.7 vbatt_max_volt

```
uint16_t vbatt_max_volt
```

Maximum battery voltage in mV.

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

- pd_common/[pd.h](#)

5.14 pd_do_t::BAT_SNK Struct Reference

Data Fields

- uint32_t [op_power](#): 10
- uint32_t [min_voltage](#): 10
- uint32_t [max_voltage](#): 10
- uint32_t [supply_type](#): 2

5.14.1 Field Documentation

5.14.1.1 max_voltage

```
uint32_t max_voltage
```

Maximum voltage in 50mV units.

5.14.1.2 min_voltage

```
uint32_t min_voltage
```

Minimum voltage in 50mV units.

5.14.1.3 op_power

```
uint32_t op_power
```

Maximum power in 250mW units.

5.14.1.4 supply_type

```
uint32_t supply_type
```

Supply type - should be 'b01.

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

- pd_common/[pd.h](#)

5.15 pd_do_t::BAT_SRC Struct Reference

Data Fields

- `uint32_t max_power: 10`
- `uint32_t min_voltage: 10`
- `uint32_t max_voltage: 10`
- `uint32_t supply_type: 2`

5.15.1 Field Documentation

5.15.1.1 max_power

`uint32_t max_power`

Maximum power in 250mW units.

5.15.1.2 max_voltage

`uint32_t max_voltage`

Maximum voltage in 50mV units.

5.15.1.3 min_voltage

`uint32_t min_voltage`

Minimum voltage in 50mV units.

5.15.1.4 supply_type

`uint32_t supply_type`

Supply type - should be 'b01.

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

- `pd_common/pd.h`

5.16 bb_handle_t Struct Reference

```
#include <billboard.h>
```

Data Fields

- `bb_type_t type`
- `bb_state_t state`
- `uint8_t num_alt_modes`
- `uint8_t bb_add_info`
- `uint32_t alt_status`
- `uint32_t timeout`

- `uint8_t * ep0_buffer`
- `bool usb_configured`
- `uint8_t usb_port`
- `bool flashing_mode`
- `bool usb_i2cm_mode`
- `bool queue_enable`
- `bool queue_disable`
- `bool queue_i2cm_enable`

5.16.1 Detailed Description

Internal billboard module handle structure.

No explicit structure for the handle is expected to be created outside of the billboard module implementation.

5.16.2 Field Documentation

5.16.2.1 alt_status

```
uint32_t alt_status
```

Current alternate mode status - The status can hold a maximum of 16 alternate modes (2bits each).

5.16.2.2 bb_add_info

```
uint8_t bb_add_info
```

AdditionalFailureInfo field in billboard capability descriptor.

5.16.2.3 ep0_buffer

```
uint8_t* ep0_buffer
```

EP0 data buffer pointer in case of internal billboard implementation.

5.16.2.4 flashing_mode

```
bool flashing_mode
```

USB flashing mode state. Valid only for internal billboard implementation.

5.16.2.5 num_alt_modes

```
uint8_t num_alt_modes
```

Number of valid alternate modes

5.16.2.6 queue_disable

```
bool queue_disable
```

Billboard interface disable request queued. Valid only for internal billboard implementation.

5.16.2.7 queue_enable

bool queue_enable

Billboard interface enable request queued. Valid only for internal billboard implementation.

5.16.2.8 queue_i2cm_enable

bool queue_i2cm_enable

USB-I2C master mode enable request. Valid only for internal billboard implementation.

5.16.2.9 state

`bb_state_t` state

Billboard current state

5.16.2.10 timeout

uint32_t timeout

Pending timeout count in ms for billboard interface disable.

5.16.2.11 type

`bb_type_t` type

Billboard implementation type

5.16.2.12 usb_configured

bool usb_configured

Internal flag indicating whether the configuration is enabled or not. This is valid only for internal billboard implementation.

5.16.2.13 usb_i2cm_mode

bool usb_i2cm_mode

USB I2C master bridge mode state. Valid only for internal billboard implementation supporting I2C master bridge interface.

5.16.2.14 usb_port

uint8_t usb_port

USB port index used in case the device has multiple USB ports.

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

- [app/billboard.h](#)

5.17 bb_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t volatile bb_enable
- uint8_t volatile bb_always_on
- uint8_t volatile bb_option
- uint16_t bb_timeout
- uint8_t volatile bb_ec_present
- uint8_t volatile bb_bus_power
- uint8_t volatile bb_unique_container_id
- uint8_t reserved_1
- uint16_t volatile bb_bos_dscr_offset
- uint16_t bb_string_dscr_offset [15]
- uint16_t bb_vid
- uint16_t bb_pid

5.17.1 Detailed Description

Struct to hold the Billboard settings.

5.17.2 Field Documentation

5.17.2.1 bb_always_on

```
uint8_t volatile bb_always_on
```

Byte 0x02: This field is valid only if the bb_enable is non-zero. The field determines when to enable the billboard interface. 0 => Enable the billboard device only on error. 1 => Always enable billboard on connection.

5.17.2.2 bb_bos_dscr_offset

```
uint16_t volatile bb_bos_dscr_offset
```

Byte 0x0A: This field is valid only for devices that have internal USB support. The field provides the offset inside the table where the BOS descriptor for the device is located. The BOS descriptor is mandatory for billboard support.

5.17.2.3 bb_bus_power

```
uint8_t volatile bb_bus_power
```

Byte 0x07: This field is valid only for devices that have internal USB support. The field indicates whether the device is bus powered or not.

5.17.2.4 bb_ec_present

```
uint8_t volatile bb_ec_present
```

Byte 0x06: EC present notification to external billboard controller.

5.17.2.5 bb_enable

```
uint8_t volatile bb_enable
```

Byte 0x01: This field indicates whether a billboard device is enabled. The usage and requirement for the billboard device is application specific. 0 => No billboard device. 1 => External billboard device. 2 => Internal billboard.

5.17.2.6 bb_option

```
uint8_t volatile bb_option
```

Byte 0x03: This field provides the various functionalities supported by the device. Bit 0 => 0 = No HID interface, 1 = Enable HID interface. Bits 1:7 => Reserved.

5.17.2.7 bb_pid

```
uint16_t bb_pid
```

Byte 0x2C: PID for the Billboard device.

5.17.2.8 bb_string_dscr_offset

```
uint16_t bb_string_dscr_offset[15]
```

Byte 0x0C: This field is valid only for devices that have internal USB support. The field provides the offset inside the table where the various string descriptors for the device are located. The descriptors are expected to be in a fixed order and is mandatory. The following are the usage models for the various string indices: 0 => Manufacturer string (mandatory). 1 => Product string (mandatory). 2 => Serial string (optional). 0000h = No serial string descriptor, FFFFh = Unique serial string generated by device, Any other offset is treated as a valid serial string. 3 => Configuration string (mandatory). 4 => Billboard interface string (mandatory). 5 => HID interface string (mandatory). 6 => Additional info URL string (mandatory). 7-8 => Alternate mode strings (mandatory for all valid modes).

5.17.2.9 bb_timeout

```
uint16_t bb_timeout
```

Byte 0x04: This field is valid only if bb_enable is non-zero. The field determines how long the billboard interface stays on, in seconds. FFFFh => Stays on until disconnect. 000Ah to FFFEh => Timeout in seconds.

5.17.2.10 bb_unique_container_id

```
uint8_t volatile bb_unique_container_id
```

Byte 0x08: This field is valid only for devices that have internal USB support. The field indicates whether the device creates the container ID descriptor or uses what is provided in the BOS descriptor.

5.17.2.11 bb_vid

```
uint16_t bb_vid
```

Byte 0x2A: VID for the Billboard device.

5.17.2.12 reserved_1

```
uint8_t reserved_1
```

Byte 0x09: Reserved area for future expansion.

5.17.2.13 table_len

```
uint8_t table_len
```

Byte 0x00: Billboard parameter table length in bytes.

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

- pd_common/pd.h

5.18 pd_do_t::BIST_DO Struct Reference

Data Fields

- uint32_t **rsvd1**: 16
- uint32_t **rsvd2**: 12
- uint32_t **mode**: 4

5.18.1 Field Documentation

5.18.1.1 mode

```
uint32_t mode
```

BIST mode.

5.18.1.2 rsvd1

```
uint32_t rsvd1
```

Reserved field.

5.18.1.3 rsvd2

```
uint32_t rsvd2
```

Reserved field.

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

- pd_common/pd.h

5.19 cc_state_t Union Reference

```
#include <pd.h>
```

Data Fields

- uint16_t **state**
- uint8_t **cc** [2]

5.19.1 Detailed Description

Union to hold CC status.

5.19.2 Field Documentation

5.19.2.1 cc

```
uint8_t cc[2]
```

Individual status of CC1(cc[0]) and CC2(cc[1]).

5.19.2.2 state

```
uint16_t state
```

Combined status of CC1 and CC2.

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

- pd_common/[pd.h](#)

5.20 ccg_timer_t Struct Reference

```
#include <timer.h>
```

Data Fields

- `uint32_t count`
- `timer_cb_t cb`
- `uint16_t period`
- `timer_id_t id`

5.20.1 Detailed Description

Structure encapsulating information relating to a software timer.

This structure encapsulates all information related to a software timer object. The timer module maintains a list of these objects corresponding to the active software timers at any given point of time.

5.20.2 Field Documentation

5.20.2.1 cb

```
timer_cb_t cb
```

The callback function pointer for the timer object.

5.20.2.2 count

```
uint32_t count
```

The actual count value.

5.20.2.3 id

```
timer_id_t id
```

The ID handle associated with the timer object.

5.20.2.4 period

```
uint16_t period
```

Period of operation for the timer instance.

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

- [system/timer.h](#)

5.21 chg_cfg_params_t Struct Reference

```
#include <pd.h>
```

Data Fields

- [uint8_t table_len](#)
- [uint8_t src_sel](#)
- [uint8_t snk_sel](#)
- [uint8_t reserved_0](#)
- [uint8_t apple_src_id](#)
- [uint8_t qc_src_type](#)
- [uint8_t reserved_1](#)
- [uint8_t afc_src_cap_cnt](#)
- [uint8_t afc_src_caps \[16\]](#)

5.21.1 Detailed Description

Struct to hold the port legacy charging parameters.

5.21.2 Field Documentation

5.21.2.1 afc_src_cap_cnt

```
uint8_t afc_src_cap_cnt
```

Number of AFC source voltage-current capabilities supported.

5.21.2.2 `afc_src_caps`

```
uint8_t afc_src_caps[16]
```

AFC source voltage-current capabilities list.

5.21.2.3 `apple_src_id`

```
uint8_t apple_src_id
```

Apple charger brick ID to be used as source: Bit 0 -> 1 A Bit 1 -> 2.1 A Bit 2 -> 2.4 A

5.21.2.4 `qc_src_type`

```
uint8_t qc_src_type
```

Quick charge source type: Bit 0 -> QC 2.0 Class-A Bit 1 -> QC 2.0 Class-B Bit 2 -> QC 3.0 Class-A Bit 3 -> QC 3.0 Class-B

5.21.2.5 `reserved_0`

```
uint8_t reserved_0
```

Reserved for future use

5.21.2.6 `reserved_1`

```
uint8_t reserved_1
```

Reserved for future use

5.21.2.7 `snk_sel`

```
uint8_t snk_sel
```

Legacy charging sink selection bit mask: Bit 0 -> BC 1.2 Bit 1 -> Apple brick Other bits reserved

5.21.2.8 `src_sel`

```
uint8_t src_sel
```

Legacy charging source selection bit mask: Bit 0 -> BC 1.2 support Bit 1 -> Apple charger Bit 2 -> Quick charge Bit 3 -> AFC charger Other bits reserved

5.21.2.9 `table_len`

```
uint8_t table_len
```

Legacy charging parameter table length in bytes

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

- [pd_common/pd.h](#)

5.22 comp_tbl_t Struct Reference

```
#include <alt_modes_mngr.h>
```

Data Fields

- uint16_t [svid](#)
- uint8_t [alt_mode_id](#)

5.22.1 Detailed Description

This structure are used in alt_modes_config.h file to set alternative modes compatibility and priority.

5.22.2 Field Documentation

5.22.2.1 alt_mode_id

```
uint8_t alt_mode_id
```

Alternative mode ID.

5.22.2.2 svrid

```
uint16_t svrid
```

Alternative mode SVID.

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

- app/alt_mode/[alt_modes_mngr.h](#)

5.23 contract_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint16_t [cur_pwr](#)
- uint16_t [max_volt](#)
- uint16_t [min_volt](#)

5.23.1 Detailed Description

Structure to hold PD Contract information.

5.23.2 Field Documentation

5.23.2.1 cur_pwr

uint16_t cur_pwr

PD contract current/power.

5.23.2.2 max_volt

uint16_t max_volt

PD contract max voltage in mV

5.23.2.3 min_volt

uint16_t min_volt

PD contract min voltage in mV

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

- pd_common/pd.h

5.24 pd_do_t::DP_CONFIG_VDO Struct Reference

Data Fields

- uint32_t [sel_conf](#): 2
- uint32_t [sign](#): 4
- uint32_t [rsvd1](#): 2
- uint32_t [dfp_asgmt](#): 8
- uint32_t [ufp_asgmt](#): 8
- uint32_t [rsvd2](#): 8

5.24.1 Field Documentation

5.24.1.1 dfp_asgmt

uint32_t dfp_asgmt

DFP_D pin assignment.

5.24.1.2 rsvd1

uint32_t rsvd1

Reserved.

5.24.1.3 rsvd2

uint32_t rsvd2

Reserved field.

5.24.1.4 sel_conf

```
uint32_t sel_conf
```

Select configuration.

5.24.1.5 sign

```
uint32_t sign
```

Signalling for DP protocol.

5.24.1.6 ufp_asgmtnt

```
uint32_t ufp_asgmtnt
```

UFP_D pin assignment.

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

- pd_common/[pd.h](#)

5.25 pd_do_t::DP_STATUS_VDO Struct Reference

Data Fields

- uint32_t [dfp_ufp_conn](#): 2
- uint32_t [pwr_low](#): 1
- uint32_t [en](#): 1
- uint32_t [mult_fun](#): 1
- uint32_t [usb_cfg](#): 1
- uint32_t [exit](#): 1
- uint32_t [hpd_state](#): 1
- uint32_t [hpd_irq](#): 1
- uint32_t [rsvd](#): 23

5.25.1 Field Documentation

5.25.1.1 dfp_ufp_conn

```
uint32_t dfp_ufp_conn
```

Whether DFP_D/UFP_D is connected.

5.25.1.2 en

```
uint32_t en
```

DP functionality enabled.

5.25.1.3 exit

```
uint32_t exit
```

Exit DP mode request.

5.25.1.4 hpd_irq

```
uint32_t hpd_irq
```

HPD IRQ status.

5.25.1.5 hpd_state

```
uint32_t hpd_state
```

Current HPD state.

5.25.1.6 mult_fun

```
uint32_t mult_fun
```

Multi-function mode preferred.

5.25.1.7 pwr_low

```
uint32_t pwr_low
```

Low power mode.

5.25.1.8 rsvd

```
uint32_t rsvd
```

Reserved field.

5.25.1.9 usb_cfg

```
uint32_t usb_cfg
```

Request switch to USB configuration.

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

- pd_common/[pd.h](#)

5.26 dpm_pd_cmd_buf_t Struct Reference

```
#include <pd.h>
```

Data Fields

- [sop_t cmd_sop](#)
- [extd_msg_t extd_type](#)
- [pd_extd_hdr_t extd_hdr](#)

- uint8_t no_of_cmd_do
- uint8_t * dat_ptr
- uint8_t timeout
- pd_do_t cmd_do [MAX_NO_OF_DO]

5.26.1 Detailed Description

Struct to hold PD command buffer.

Warning

When providing pointer to the extended data make sure original buffer is always 4-byte aligned. i.e, even if 1 byte data is required, 4 bytes should be used to store that data.

5.26.2 Field Documentation

5.26.2.1 cmd_do

pd_do_t cmd_do [MAX_NO_OF_DO]

Command data objects.

5.26.2.2 cmd_sop

sop_t cmd_sop

SOP type

5.26.2.3 dat_ptr

uint8_t* dat_ptr

Data Pointer in case of extended message only

5.26.2.4 extd_hdr

pd_extd_hdr_t extd_hdr

Extended Header

5.26.2.5 extd_type

extd_msg_t extd_type

Extended Message Type

5.26.2.6 no_of_cmd_do

uint8_t no_of_cmd_do

No of data objects including VDM header

5.26.2.7 timeout

```
uint8_t timeout
```

Timeout value, in ms, for a response. If set to zero, the PD stack will not wait for a VDM response and jump to ready state after this buffer has been sent.

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

- pd_common/pd.h

5.27 dpm_status_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t port_role
- uint8_t dfilt_port_role
- uint8_t src_cur_level
- uint8_t is_src_bat
- uint8_t is_snk_bat
- uint8_t snk_usb_susp_en
- uint8_t snk_usb_comm_en
- uint8_t swap_response
- uint8_t toggle
- uint8_t src_pdo_count
- uint8_t src_pdo_mask
- uint8_t snk_pdo_count
- uint8_t snk_pdo_mask
- uint8_t rp_supported
- uint8_t pd_support
- uint8_t try_src_snk
- uint8_t cbl_dsc
- uint8_t db_support
- uint8_t err_recov
- uint8_t port_disable
- uint8_t frs_enable
- uint8_t vconn_retain
- uint16_t reserved_3 [5]
- pd_do_t src_pdo [MAX_NO_OF_PDO]
- pd_do_t snk_pdo [MAX_NO_OF_PDO]
- uint16_t snk_max_min [MAX_NO_OF_PDO]
- port_type_t cur_port_type
- port_role_t cur_port_role
- uint8_t volatile snk_cur_level
- uint8_t volatile bootup
- uint8_t volatile dead_bat
- uint8_t drp_period
- uint8_t src_period
- uint8_t snk_period
- uint8_t volatile skip_scan
- uint8_t polarity
- uint8_t rev_pol

- uint8_t volatile `connect`
- uint8_t volatile `attach`
- `port_role_t role_at_connect`
- `pd_devtype_t attached_dev`
- uint8_t volatile `contract_exist`
- uint8_t volatile `pd_connected`
- uint8_t volatile `pd_disabled`
- uint8_t volatile `ra_present`
- uint8_t volatile `emca_present`
- uint8_t volatile `bist_cm2_enabled`
- `std_vdm_prod_t cbl_type`
- `std_vdm_ver_t cbl_vdm_version`
- uint8_t volatile `vconn_logical`
- uint8_t `cur_src_pdo_count`
- uint8_t `cur_snk_pdo_count`
- uint8_t `cbl_wait`
- `pe_cbl_state_t cbl_state`
- uint8_t `cbl_soft_reset_tried`
- `typec_fsm_state_t typec_fsm_state`
- `dpm_pd_cmd_t dpm_pd_cmd`
- uint8_t `dpm_pd_cmd_active`
- uint8_t `dpm_typec_cmd_active`
- bool `dpm_enabled`
- bool `dpm_init`
- uint8_t `dpm_safe_disable`
- `dpm_typec_cmd_t dpm_typec_cmd`
- uint8_t `src_cur_level_live`
- `cc_state_t cc_live`
- `cc_state_t cc_status`
- `cc_state_t cc_old_status`
- `cc_state_t cc_rd_status`
- `pd_rev_t spec_rev_sop_live`
- `pd_rev_t spec_rev_sop_prime_live`
- `pd_rev_t spec_rev_cbl`
- `pd_rev_t spec_rev_peer`
- bool `unchunk_sup_live`
- bool `unchunk_sup_peer`
- bool `snk_rp_detach_en`
- bool `cur_fb`
- `pd_ams_type non_intr_response`
- bool `fr_rx_disabled`
- bool `fr_rx_en_live`
- bool `fr_tx_disabled`
- bool `fr_tx_en_live`
- volatile bool `fault_active`
- `pe_fsm_state_t pe_fsm_state`
- uint32_t volatile `pe_evt`
- uint32_t volatile `typec_evt`
- `contract_t contract`
- `pd_do_t alert`
- `pd_do_t cbl_vdo`
- bool `cbl_mode_en`
- `app_cbk_t * app_cbk`
- `dpm_pd_cmd_cbk_t dpm_pd_cbk`
- `dpm_typec_cmd_cbk_t dpm_typec_cbk`

- `dpm_pd_cmd_buf_t * cmd_p`
- `dpm_pd_cmd_buf_t dpm_cmd_buf`
- `uint16_t cur_snk_max_min [MAX_NO_OF PDO]`
- `pd_do_t cur_src_pdo [MAX_NO_OF PDO]`
- `pd_do_t cur_snk_pdo [MAX_NO_OF PDO]`
- `pd_do_t src_cur_rdo`
- `pd_do_t src_last_rdo`
- `pd_do_t src_rdo`
- `pd_do_t snk_rdo`
- `pd_do_t snk_sel_pdo`
- `pd_do_t src_sel_pdo`
- `pd_packet_t * src_cap_p`
- `uint32_t padval`
- `pd_power_status_t port_status`
- `uint8_t ext_src_cap [CCG_PD_EXT_SRCCAP_SIZE]`
- `uint8_t pps_status [CCG_PD_EXT_PPS_STATUS_SIZE]`
- `uint8_t dpm_err_info`

5.27.1 Detailed Description

PD Device Policy Status structure. This structure holds all of the configuration and status information associated with a port on the CCG device. Members of this structure should not be directly modified by any of the application code.

Warning

Initial elements of this structure maps directly to config table fields and hence must not be moved around or changed.

5.27.2 Field Documentation

5.27.2.1 alert

`pd_do_t alert`

Alert status

5.27.2.2 app_cbk

`app_cbk_t* app_cbk`

Application callback pointer.

5.27.2.3 attach

`uint8_t volatile attach`

Port attached indication.

5.27.2.4 attached_dev

`pd_devtype_t attached_dev`

Type of device attached.

5.27.2.5 bist_cm2_enabled

```
uint8_t volatile bist_cm2_enabled
```

BIST Carrier Mode 2 going on

5.27.2.6 bootup

```
uint8_t volatile bootup
```

Flag to indicate chip bootup, used to check dead battery.

5.27.2.7 cbl_dsc

```
uint8_t cbl_dsc
```

Cable discovery control knob.

5.27.2.8 cbl_mode_en

```
bool cbl_mode_en
```

Whether cable supports alternate modes.

5.27.2.9 cbl_soft_reset_tried

```
uint8_t cbl_soft_reset_tried
```

Stores number of cable soft reset tries.

5.27.2.10 cbl_state

```
pe_cbl_state_t cbl_state
```

Cable discovery state machine state.

5.27.2.11 cbl_type

```
std_vdm_prod_t cbl_type
```

Stores the cable type.

5.27.2.12 cbl_vdm_version

```
std_vdm_ver_t cbl_vdm_version
```

Stores the cable VDM version.

5.27.2.13 cbl_vdo

```
pd_do_t cbl_vdo
```

Stores the last received cable VDO.

5.27.2.14 cbl_wait

```
uint8_t cbl_wait
```

Flag to indicate cable discovery is waiting for some event.

5.27.2.15 cc_live

```
cc_state_t cc_live
```

Live CC status.

5.27.2.16 cc_old_status

```
cc_state_t cc_old_status
```

Old CC status.

5.27.2.17 cc_rd_status

```
cc_state_t cc_rd_status
```

Rd status.

5.27.2.18 cc_status

```
cc_state_t cc_status
```

Current debounced CC status.

5.27.2.19 cmd_p

```
dpm_pd_cmd_buf_t* cmd_p
```

Pointer to DPM command buffer.

5.27.2.20 connect

```
uint8_t volatile connect
```

Port connected but not debounced yet.

5.27.2.21 contract

```
contract_t contract
```

Current pd contract.

5.27.2.22 contract_exist

```
uint8_t volatile contract_exist
```

PD contract exists indication.

5.27.2.23 cur_fb

`bool cur_fb`

Flag to indicate current foldback is active

5.27.2.24 cur_port_role

`port_role_t cur_port_role`

Current Port role: Sink or Source.

5.27.2.25 cur_port_type

`port_type_t cur_port_type`

Current port type: UFP or DFP.

5.27.2.26 cur_snk_max_min

`uint16_t cur_snk_max_min[MAX_NO_OF_PDO]`

Max min current/power of current sink capabilities.

5.27.2.27 cur_snk_pdo

`pd_do_t cur_snk_pdo[MAX_NO_OF_PDO]`

Current sink PDOs sent in sink cap messages.

5.27.2.28 cur_snk_pdo_count

`uint8_t cur_snk_pdo_count`

Sink PDO count in the last sent sink cap.

5.27.2.29 cur_src_pdo

`pd_do_t cur_src_pdo[MAX_NO_OF_PDO]`

Current source PDOs sent in source cap messages.

5.27.2.30 cur_src_pdo_count

`uint8_t cur_src_pdo_count`

Source PDO count in the last sent source cap.

5.27.2.31 db_support

`uint8_t db_support`

Dead battery support control knob.

5.27.2.32 dead_bat

```
uint8_t volatile dead_bat
```

Flag to indicate dead battery operation.

5.27.2.33 dfilt_port_role

```
uint8_t dfilt_port_role
```

Default port role: Sink or Source.

5.27.2.34 dpm_cmd_buf

```
dpm_pd_cmd_buf_t dpm_cmd_buf
```

Local DPM command buffer.

5.27.2.35 dpm_enabled

```
bool dpm_enabled
```

DPM enabled flag.

5.27.2.36 dpm_err_info

```
uint8_t dpm_err_info
```

Additional error status for DPM commands.

5.27.2.37 dpm_init

```
bool dpm_init
```

DPM Initialized flag.

5.27.2.38 dpm_pd_cbk

```
dpm_pd_cmd_cbk_t dpm_pd_cbk
```

Pointer to DPM PD callback function.

5.27.2.39 dpm_pd_cmd

```
dpm_pd_cmd_t dpm_pd_cmd
```

Current DPM PD command.

5.27.2.40 dpm_pd_cmd_active

```
uint8_t dpm_pd_cmd_active
```

Indicate DPM PD command was registered.

5.27.2.41 dpm_safe_disable

```
uint8_t dpm_safe_disable
```

DPM sage disable flag. Used to make sure that the port is disabled correctly after a fault occurrence.

5.27.2.42 dpm_typec_cbk

```
dpm_typec_cmd_cbk_t dpm_typec_cbk
```

Pointer to DPM Type C callback function.

5.27.2.43 dpm_typec_cmd

```
dpm_typec_cmd_t dpm_typec_cmd
```

Current DPM Type C command.

5.27.2.44 dpm_typec_cmd_active

```
uint8_t dpm_typec_cmd_active
```

Indicate DPM Type C command was registered.

5.27.2.45 drp_period

```
uint8_t drp_period
```

Time period for DRP toggling.

5.27.2.46 emca_present

```
uint8_t volatile emca_present
```

EMCA cable present indication.

5.27.2.47 err_recov

```
uint8_t err_recov
```

Error recovery control knob.

5.27.2.48 ext_src_cap

```
uint8_t ext_src_cap[CCG_PD_EXT_SRCCAP_SIZE]
```

Buffer to hold extended source caps.

5.27.2.49 fault_active

```
volatile bool fault_active
```

Flag to indicate the a fault condition exists.

5.27.2.50 fr_rx_disabled

bool fr_rx_disabled

FRS receive disabled by EC.

5.27.2.51 fr_rx_en_live

bool fr_rx_en_live

Fast role signal detection enabled

5.27.2.52 fr_tx_disabled

bool fr_tx_disabled

FRS transmit disabled by EC.

5.27.2.53 fr_tx_en_live

bool fr_tx_en_live

Fast role signal auto send enabled

5.27.2.54 frs_enable

uint8_t frs_enable

FRS enable flags.

5.27.2.55 is_snk_bat

uint8_t is_snk_bat

Power sink is connected to a battery.

5.27.2.56 is_src_bat

uint8_t is_src_bat

Power source is connected to a battery.

5.27.2.57 non_intr_response

pd_ams_type non_intr_response

Flag to indicate stack is waiting for App response to a non interruptible AMS

5.27.2.58 padval

uint32_t padval

Fields below need to be properly aligned to 4 byte boundary

5.27.2.59 pd_connected

```
uint8_t volatile pd_connected
```

Ports are PD connected indication.

5.27.2.60 pd_disabled

```
uint8_t volatile pd_disabled
```

PD disabled indication.

5.27.2.61 pd_support

```
uint8_t pd_support
```

USB-PD supported.

5.27.2.62 pe_evt

```
uint32_t volatile pe_evt
```

Stores policy engine events.

5.27.2.63 pe_fsm_state

```
pe_fsm_state_t pe_fsm_state
```

Holds the current state of Policy Engine (PE).

5.27.2.64 polarity

```
uint8_t polarity
```

CC channel polarity (CC1=0, CC2=1)

5.27.2.65 port_disable

```
uint8_t port_disable
```

PD port disable flag.

5.27.2.66 port_role

```
uint8_t port_role
```

Port role: Sink, Source or Dual.

5.27.2.67 port_status

```
pd_power_status_t port_status
```

Port power status.

5.27.2.68 pps_status

```
uint8_t pps_status[CCG_PD_EXT_PPS_STATUS_SIZE]
```

Buffer to hold PPS status.

5.27.2.69 ra_present

```
uint8_t volatile ra_present
```

Ra present indication.

5.27.2.70 reserved_3

```
uint16_t reserved_3[5]
```

Reserved words for padding to 4-byte aligned address.

5.27.2.71 rev_pol

```
uint8_t rev_pol
```

CC channel reverse polarity.

5.27.2.72 role_at_connect

```
port_role_t role_at_connect
```

Port role when the port moved to the attached state.

5.27.2.73 rp_supported

```
uint8_t rp_supported
```

Supported Rp values bit mask.

- Bit 0 => Default Rp supported.
- Bit 1 => 1.5 A Rp supported.
- Bit 2 => 3 A Rp supported.

5.27.2.74 skip_scan

```
uint8_t volatile skip_scan
```

Skip CC scanning control knob.

5.27.2.75 snk_cur_level

```
uint8_t volatile snk_cur_level
```

Type C current level in sink role.

5.27.2.76 snk_max_min

```
uint16_t snk_max_min[MAX_NO_OF_PDO]
```

Max min current from the config table or updated at runtime by the EC.

5.27.2.77 snk_pdo

```
pd_do_t snk_pdo[MAX_NO_OF_PDO]
```

Sink PDO loaded from the config table or updated at runtime by the EC.

5.27.2.78 snk_pdo_count

```
uint8_t snk_pdo_count
```

Sink PDO count from the config table or updated at runtime by the EC.

5.27.2.79 snk_pdo_mask

```
uint8_t snk_pdo_mask
```

Sink PDO mask from the config table or updated at runtime by the EC.

5.27.2.80 snk_period

```
uint8_t snk_period
```

Time period for which to stay as a SNK for a DRP device.

5.27.2.81 snk_rdo

```
pd_do_t snk_rdo
```

Last RDO sent (when operating as a sink) that resulted in a contract.

5.27.2.82 snk_rp_detach_en

```
bool snk_rp_detach_en
```

Flag to indicate sink will detach on Rp removal instead of VBUS removal.

5.27.2.83 snk_sel_pdo

```
pd_do_t snk_sel_pdo
```

Selected PDO which resulted in contract (when sink).

5.27.2.84 snk_usb_comm_en

```
uint8_t snk_usb_comm_en
```

USB communication supported indication.

5.27.2.85 snk_usb_susp_en

`uint8_t snk_usb_susp_en`

USB suspend supported indication.

5.27.2.86 spec_rev_cbl

`pd_rev_t spec_rev_cbl`

Spec revision of the currently connected cable.

5.27.2.87 spec_rev_peer

`pd_rev_t spec_rev_peer`

Spec revision of the currently connected peer.

5.27.2.88 spec_rev_sop_live

`pd_rev_t spec_rev_sop_live`

Current spec rev for SOP messages.

5.27.2.89 spec_rev_sop_prime_live

`pd_rev_t spec_rev_sop_prime_live`

Current spec revision for SOP Prime/DPrime messages.

5.27.2.90 src_cap_p

`pd_packet_t* src_cap_p`

Pointer to the current/last source cap message received. May be NULL. Data pointed to by this should not be changed.

5.27.2.91 src_cur_level

`uint8_t src_cur_level`

Type C current level in the source role.

5.27.2.92 src_cur_level_live

`uint8_t src_cur_level_live`

Current Type C current level in the source role.

5.27.2.93 src_cur_rdo

`pd_do_t src_cur_rdo`

Stores the current rdo received by source

5.27.2.94 src_last_rdo

`pd_do_t src_last_rdo`

Stores the last rdo received by source

5.27.2.95 src_pdo

`pd_do_t src_pdo[MAX_NO_OF_PDO]`

Source PDO loaded from the config table or updated at runtime by the EC.

5.27.2.96 src_pdo_count

`uint8_t src_pdo_count`

Source PDO count from the config table or updated at runtime by the EC.

5.27.2.97 src_pdo_mask

`uint8_t src_pdo_mask`

Source PDO mask from the config table or updated at runtime by the EC.

5.27.2.98 src_period

`uint8_t src_period`

Time period for which to stay as a SRC for a DRP device.

5.27.2.99 src_rdo

`pd_do_t src_rdo`

Last RDO received (when operating as a source) that resulted in a contract.

5.27.2.100 src_sel_pdo

`pd_do_t src_sel_pdo`

Selected PDO which resulted in contract (when source).

5.27.2.101 swap_response

`uint8_t swap_response`

Response to be sent for each USB-PD SWAP command.

- Bits 1:0 => DR_SWAP response.
- Bits 3:2 => PR_SWAP response.
- Bits 5:4 => VCONN_SWAP response. Allowed values are: 0=ACCEPT, 1=REJECT, 2=WAIT, 3=NOT_SUPPORTED.

5.27.2.102 toggle

```
uint8_t toggle
```

DRP toggle is enabled.

5.27.2.103 try_src_snk

```
uint8_t try_src_snk
```

Try Source/ Try Sink control knob.

5.27.2.104 typec_evt

```
uint32_t volatile typec_evt
```

Stores Type C events.

5.27.2.105 typec_fsm_state

```
typec_fsm_state_t typec_fsm_state
```

Type C generic FSM state.

5.27.2.106 unchunk_sup_live

```
bool unchunk_sup_live
```

Mutual unchunk support with the currently connected peer.

5.27.2.107 unchunk_sup_peer

```
bool unchunk_sup_peer
```

Unchunk support of the currently connected peer.

5.27.2.108 vconn_logical

```
uint8_t volatile vconn_logical
```

VCONN logical status.

5.27.2.109 vconn_retain

```
uint8_t vconn_retain
```

Whether VConn should be retained in ON state.

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

- pd_common/[pd.h](#)

5.28 pd_do_t::FIXED_SNK Struct Reference

Data Fields

- `uint32_t op_current: 10`
- `uint32_t voltage: 10`
- `uint32_t rsrvd: 3`
- `uint32_t fr_swap: 2`
- `uint32_t dr_swap: 1`
- `uint32_t usb_comm_cap: 1`
- `uint32_t extPowered: 1`
- `uint32_t high_cap: 1`
- `uint32_t dual_role_power: 1`
- `uint32_t supply_type: 2`

5.28.1 Field Documentation

5.28.1.1 dr_swap

`uint32_t dr_swap`

Data Role Swap supported.

5.28.1.2 dual_role_power

`uint32_t dual_role_power`

Dual role power support.

5.28.1.3 extPowered

`uint32_t extPowered`

Externally powered.

5.28.1.4 fr_swap

`uint32_t fr_swap`

FR swap support.

5.28.1.5 high_cap

`uint32_t high_cap`

Higher capability possible.

5.28.1.6 op_current

`uint32_t op_current`

Operational current in 10mA units.

5.28.1.7 rsrvd

uint32_t rsrvd

Reserved field.

5.28.1.8 supply_type

uint32_t supply_type

Supply type - should be 'b00.

5.28.1.9 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

5.28.1.10 voltage

uint32_t voltage

Voltage in 50mV units.

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

- pd_common/[pd.h](#)

5.29 pd_do_t::FIXED_SRC Struct Reference

Data Fields

- uint32_t [max_current](#): 10
- uint32_t [voltage](#): 10
- uint32_t [pk_current](#): 2
- uint32_t [reserved](#): 2
- uint32_t [unchunk_sup](#): 1
- uint32_t [dr_swap](#): 1
- uint32_t [usb_comm_cap](#): 1
- uint32_t [extPowered](#): 1
- uint32_t [usb_suspend_sup](#): 1
- uint32_t [dual_role_power](#): 1
- uint32_t [supply_type](#): 2

5.29.1 Field Documentation

5.29.1.1 dr_swap

uint32_t dr_swap

Data Role Swap supported.

5.29.1.2 dual_role_power

```
uint32_t dual_role_power
```

Dual role power support.

5.29.1.3 extPowered

```
uint32_t extPowered
```

Externally powered.

5.29.1.4 max_current

```
uint32_t max_current
```

Maximum current in 100mA units.

5.29.1.5 pk_current

```
uint32_t pk_current
```

Peak current.

5.29.1.6 reserved

```
uint32_t reserved
```

Reserved field.

5.29.1.7 supply_type

```
uint32_t supply_type
```

Supply type - should be 'b00.

5.29.1.8 unchunk_sup

```
uint32_t unchunk_sup
```

Unchunked extended messages supported.

5.29.1.9 usb_comm_cap

```
uint32_t usbCommCap
```

USB communication capability.

5.29.1.10 usb_suspend_sup

```
uint32_t usbSuspendSup
```

USB suspend supported.

5.29.1.11 voltage

```
uint32_t voltage
```

Voltage in 50mV units.

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

- pd_common/pd.h

5.30 fw_img_status_t Union Reference

```
#include <boot.h>
```

Data Structures

- struct [fw_mode_reason_t](#)

Data Fields

- uint8_t val
- struct [fw_img_status_t::fw_mode_reason_t](#) status

5.30.1 Detailed Description

Boot mode reason structure.

This structure holds status of FW images and boot mode request. If the CCGx device is running in Boot-loader mode, this register can be used to identify the reason for this. The register will report the validity of FW1 and FW2 binaries even in the case where the device is already running in FW1 or FW2 mode.

5.30.2 Field Documentation

5.30.2.1 status

```
struct fw_img_status_t::fw_mode_reason_t status
```

Union containing the status fields in the boot mode reason value.

5.30.2.2 val

```
uint8_t val
```

Integer field used for direct manipulation of reason code.

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

- system/boot.h

5.31 fw_img_status_t::fw_mode_reason_t Struct Reference

Data Fields

- uint8_t boot_mode_request: 1
- uint8_t reserved: 1
- uint8_t fw1_invalid: 1
- uint8_t fw2_invalid: 1
- uint8_t reserved1: 4

5.31.1 Field Documentation

5.31.1.1 boot_mode_request

uint8_t boot_mode_request

Boot mode request made by FW.

5.31.1.2 fw1_invalid

uint8_t fw1_invalid

FW1 image invalid: 0=Valid, 1=Invalid.

5.31.1.3 fw2_invalid

uint8_t fw2_invalid

FW2 image invalid: 0=Valid, 1=Invalid.

5.31.1.4 reserved

uint8_t reserved

Reserved field: Will be zero.

5.31.1.5 reserved1

uint8_t reserved1

Reserved for later use.

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

- system/boot.h

5.32 i2c_scb_config_t Struct Reference

```
#include <i2c.h>
```

Data Fields

- `i2c_scb_mode_t mode`
- `uint8_t slave_address`
- `uint8_t slave_mask`
- `i2c_scb_clock_freq_t clock_freq`
- `i2c_scb_state_t i2c_state`
- `uint8_t * buffer`
- `uint16_t buf_size`
- `i2c_cb_fun_t cb_fun_ptr`
- `volatile uint16_t i2c_write_count`

5.32.1 Detailed Description

This structure holds all configuration associated with each I2C block.

This structure is used internally by the driver, and is not expected to be manipulated directly by the caller.

5.32.2 Field Documentation

5.32.2.1 buf_size

`uint16_t buf_size`

Size of receive buffer.

5.32.2.2 buffer

`uint8_t* buffer`

Buffer to be used to receive data.

5.32.2.3 cb_fun_ptr

`i2c_cb_fun_t cb_fun_ptr`

Callback function pointer.

5.32.2.4 clock_freq

`i2c_scb_clock_freq_t clock_freq`

Clock frequency. Only valid in master mode.

5.32.2.5 i2c_state

`i2c_scb_state_t i2c_state`

Current state of the I2C block.

5.32.2.6 i2c_write_count

```
volatile uint16_t i2c_write_count
```

Current index into receive buffer.

5.32.2.7 mode

```
i2c_scb_mode_t mode
```

I2C operating mode.

5.32.2.8 slave_address

```
uint8_t slave_address
```

Slave address. Only valid in slave modes.

5.32.2.9 slave_mask

```
uint8_t slave_mask
```

Slave address mask. Only valid in slave modes.

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

- scb/i2c.h

5.33 ocp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t threshold
- uint8_t debounce
- uint8_t retry_cnt
- uint8_t threshold2
- uint8_t debounce2
- uint8_t sense_res
- uint8_t cs_res

5.33.1 Detailed Description

Struct to hold the OCP settings.

5.33.2 Field Documentation

5.33.2.1 cs_res

```
uint8_t cs_res
```

Current sense tuning resistor impedance in 100 Ohm units.

5.33.2.2 debounce

```
uint8_t debounce
```

OCP debounce period in milliseconds. OCP handling is only performed if the OCP condition persists for this duration.

5.33.2.3 debounce2

```
uint8_t debounce2
```

Debounce period in ms corresponding to secondary threshold.

5.33.2.4 retry_cnt

```
uint8_t retry_cnt
```

Number of consecutive OCP events allowed before the port is suspended by CCG firmware.

5.33.2.5 sense_res

```
uint8_t sense_res
```

Sense Resistor impedance in milli-Ohm units.

5.33.2.6 table_len

```
uint8_t table_len
```

Table length in bytes

5.33.2.7 threshold

```
uint8_t threshold
```

OCP threshold: Excess current in percentage of maximum allowed current.

5.33.2.8 threshold2

```
uint8_t threshold2
```

Secondary OCP threshold (corresponding to peak current condition). Set to 0 if not used.

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

- pd_common/[pd.h](#)

5.34 otp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- `uint8_t table_len`
- `uint8_t therm_type`
- `uint16_t cutoff_volt`
- `uint16_t restart_volt`
- `uint16_t debounce`
- `uint8_t reserved_0 [4]`

5.34.1 Detailed Description

Struct to hold the OTP settings.

5.34.2 Field Documentation

5.34.2.1 cutoff_volt

`uint16_t cutoff_volt`

Voltage corresponding to the temperature at which OTP cutoff is to be performed (in mV)

5.34.2.2 debounce

`uint16_t debounce`

OTP debounce duration in ms.

5.34.2.3 reserved_0

`uint8_t reserved_0 [4]`

Reserved for future use

5.34.2.4 restart_volt

`uint16_t restart_volt`

Voltage corresponding to the temperature at which system operation can be resumed (in mV)

5.34.2.5 table_len

`uint8_t table_len`

Table length in bytes

5.34.2.6 therm_type

`uint8_t therm_type`

Type of thermistor used for temperature sensing: 0 = Negative Temperature Coefficient (NTC) 1 = Positive Temperature Coefficient

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

- pd_common/[pd.h](#)

5.35 ovp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t threshold
- uint8_t debounce
- uint8_t retry_cnt

5.35.1 Detailed Description

Struct to hold the OVP settings.

5.35.2 Field Documentation

5.35.2.1 debounce

```
uint8_t debounce
```

OVP debounce duration in micro-seconds. If a non-zero debounce is specified, there can be an error of upto 35 us due to the device being in sleep mode.

5.35.2.2 retry_cnt

```
uint8_t retry_cnt
```

Number of consecutive OVP events allowed before the port operation is suspended by CCG firmware.

5.35.2.3 table_len

```
uint8_t table_len
```

Table length in bytes

5.35.2.4 threshold

```
uint8_t threshold
```

OVP threshold: Excess voltage above expected value in percentage of expected voltage

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

- pd_common/[pd.h](#)

5.36 pd_do_t::PAS_CBL_VDO Struct Reference

Data Fields

- `uint32_t usb_ss_sup`: 3
- `uint32_t rsvd1`: 2
- `uint32_t vbus_cur`: 2
- `uint32_t rsvd2`: 2
- `uint32_t max_vbus_volt`: 2
- `uint32_t缆线_term`: 2
- `uint32_t缆线_latency`: 4
- `uint32_t typec_plug`: 1
- `uint32_t typec_abc`: 2
- `uint32_t rsvd3`: 1
- `uint32_t vdo_version`: 3
- `uint32_t缆线_fw_ver`: 4
- `uint32_t缆线_hw_ver`: 4

5.36.1 Field Documentation

5.36.1.1 `cbl_fw_ver`

`uint32_t缆线_fw_ver`

Cable firmware version.

5.36.1.2 `cbl_hw_ver`

`uint32_t缆线_hw_ver`

Cable hardware version.

5.36.1.3 `cbl_latency`

`uint32_t缆线_latency`

Cable latency.

5.36.1.4 `cbl_term`

`uint32_t缆线_term`

Cable termination and VConn power requirement.

5.36.1.5 `max_vbus_volt`

`uint32_t max_vbus_volt`

Max. VBus voltage supported.

5.36.1.6 rsvd1

uint32_t rsvd1

Reserved field.

5.36.1.7 rsvd2

uint32_t rsvd2

Reserved field.

5.36.1.8 rsvd3

uint32_t rsvd3

Reserved field.

5.36.1.9 typec_abc

uint32_t typec_abc

Cable plug type.

5.36.1.10 typec_plug

uint32_t typec_plug

Reserved field.

5.36.1.11 usb_ss_sup

uint32_t usb_ss_sup

USB signalling supported by the cable.

5.36.1.12 vbus_cur

uint32_t vbus_cur

VBus current supported by the cable.

5.36.1.13 vdo_version

uint32_t vdo_version

VDO version.

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

- pd_common/[pd.h](#)

5.37 pd_config_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint16_t `table_sign`
- uint8_t `table_type`
- uint8_t `application`
- uint16_t `table_version`
- uint16_t `table_size`
- uint8_t `table_checksum`
- uint8_t `flash_checksum`
- uint16_t `flashing_vid`
- uint16_t `flashing_mode`
- uint8_t `pd_port_cnt`
- uint8_t `reserved_0`
- uint16_t `mfg_info_offset`
- uint8_t `mfg_info_length`
- uint8_t `reserved_1`
- uint16_t `reserved_2` [6]
- `pd_port_config_t port_conf` [NO_OF_TYPEC_PORTS]

5.37.1 Detailed Description

Struct to hold the PD device configuration.

5.37.2 Field Documentation

5.37.2.1 application

`uint8_t application`

Byte 0x03: This field specifies the type of PD application supported: 0 => Notebook 1 => Tablet 2 => Passive Cable 3 => Active Cable 4 => Monitor 5 => Power Adapter 6 => Cable Adapter (Dongle)

5.37.2.2 flash_checksum

`uint8_t flash_checksum`

Byte 0x09: One byte flash checksum. Used to ensure that binary sum of config table is 0.

5.37.2.3 flashing_mode

`uint16_t flashing_mode`

Byte 0x0C: Special Mode used for flashing in case of CC firmware updates.

5.37.2.4 flashing_vid

`uint16_t flashing_vid`

Byte 0x0A: Special VID used for flashing mode in case of CC firmware updates.

5.37.2.5 mfg_info_length

```
uint8_t mfg_info_length
```

Byte 0x12: Length of manufacturer information in config table.

5.37.2.6 mfg_info_offset

```
uint16_t mfg_info_offset
```

Byte 0x10: Offset to manufacturer information in config table.

5.37.2.7 pd_port_cnt

```
uint8_t pd_port_cnt
```

Byte 0x0E: Number of PD ports enabled in the configuration.

5.37.2.8 port_conf

```
pd_port_config_t port_conf[NO_OF_TYPEC_PORTS]
```

Byte 0x20 (0x110): Configuration data for each USB-PD port.

5.37.2.9 reserved_0

```
uint8_t reserved_0
```

Byte 0x0F: Reserved byte for 2-byte alignment.

5.37.2.10 reserved_1

```
uint8_t reserved_1
```

Byte 0x13: Reserved field for 2-byte alignment.

5.37.2.11 reserved_2

```
uint16_t reserved_2[6]
```

Byte 0x14: Reserved fields for future expansion.

5.37.2.12 table_checksum

```
uint8_t table_checksum
```

Byte 0x08: One byte configuration checksum. Calculated over bytes 10 to byte (size - 1).

5.37.2.13 table_sign

```
uint16_t table_sign
```

Byte 0x00: Two byte signature to indicate validity of the configuration.

5.37.2.14 table_size

```
uint16_t table_size
```

Byte 0x06: Size of the configuration table. Checksum is calculated over bytes 10 to size - 1.

5.37.2.15 table_type

```
uint8_t table_type
```

Byte 0x02: The table type indicates the type of solution. 1 => EMCA 2 => DFP 3 => UFP 4 => DRP

5.37.2.16 table_version

```
uint16_t table_version
```

Byte 0x04: Table version: This contains 4 bit major version, 4 bit minor version and 8 bit patch number.

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

- [pd_common/pd.h](#)

5.38 pd_contract_info_t Struct Reference

```
#include <pd.h>
```

Data Fields

- [pd_do_t rdo](#)
- [pd_contract_status_t status](#)

5.38.1 Detailed Description

Structure to hold PD Contract information passed with APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE event to the application.

5.38.2 Field Documentation

5.38.2.1 rdo

```
pd_do_t rdo
```

RDO associated with the contract.

5.38.2.2 status

```
pd_contract_status_t status
```

Status of the contract.

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

- [pd_common/pd.h](#)

5.39 pd_do_t Union Reference

```
#include <pd.h>
```

Data Structures

- struct [ACT_CBL_VDO](#)
- struct [BAT_SNK](#)
- struct [BAT_SRC](#)
- struct [BIST_DO](#)
- struct [DP_CONFIG_VDO](#)
- struct [DP_STATUS_VDO](#)
- struct [FIXED_SNK](#)
- struct [FIXED_SRC](#)
- struct [PAS_CBL_VDO](#)
- struct [QC_4_0_DATA_VDO](#)
- struct [RDO_BAT](#)
- struct [RDO_BAT_GIVEBACK](#)
- struct [RDO_FIXED_VAR](#)
- struct [RDO_FIXED_VAR_GIVEBACK](#)
- struct [RDO_GEN](#)
- struct [RDO_GEN_GVB](#)
- struct [SRC_GEN](#)
- struct [STD_AMA_VDO](#)
- struct [STD_AMA_VDO_PD3](#)
- struct [STD_CBL_VDO](#)
- struct [STD_CERT_VDO](#)
- struct [STD_DP_VDO](#)
- struct [STD_PROD_VDO](#)
- struct [STD_SVID_RESP_VDO](#)
- struct [STD_VDM_HDR](#)
- struct [STD_VDM_ID_HDR](#)
- struct [USTD_QC_4_0_HDR](#)
- struct [USTD_VDM_HDR](#)
- struct [VAR_SNK](#)
- struct [VAR_SRC](#)

Data Fields

- uint32_t [val](#)
- struct [pd_do_t::BIST_DO](#) [bist_do](#)
- struct [pd_do_t::FIXED_SRC](#) [fixed_src](#)
- struct [pd_do_t::VAR_SRC](#) [var_src](#)
- struct [pd_do_t::BAT_SRC](#) [bat_src](#)
- struct [pd_do_t::SRC_GEN](#) [src_gen](#)
- struct [pd_do_t::FIXED_SNK](#) [fixed_snk](#)
- struct [pd_do_t::VAR_SNK](#) [var_snk](#)
- struct [pd_do_t::BAT_SNK](#) [bat_snk](#)
- struct [pd_do_t::RDO_FIXED_VAR](#) [rdo_fix_var](#)
- struct [pd_do_t::RDO_FIXED_VAR_GIVEBACK](#) [rdo_fix_var_gvb](#)
- struct [pd_do_t::RDO_BAT](#) [rdo_bat](#)
- struct [pd_do_t::RDO_BAT_GIVEBACK](#) [rdo_bat_gvb](#)
- struct [pd_do_t::RDO_GEN](#) [rdo_gen](#)

- struct [pd_do_t::RDO_GEN_GVB](#) rdo_gen_gvb
- struct [pd_do_t::STD_VDM_HDR](#) std_vdm_hdr
- struct [pd_do_t::USTD_VDM_HDR](#) ustd_vdm_hdr
- struct [pd_do_t::USTD_QC_4_0_HDR](#) ustd_qc_4_0_hdr
- struct [pd_do_t::QC_4_0_DATA_VDO](#) qc_4_0_data_vdo
- struct [pd_do_t::STD_VDM_ID_HDR](#) std_id_hdr
- struct [pd_do_t::STD_CERT_VDO](#) std_cert_vdo
- struct [pd_do_t::STD_PROD_VDO](#) std_prod_vdo
- struct [pd_do_t::STD_CBL_VDO](#) std_cbl_vdo
- struct [pd_do_t::PAS_CBL_VDO](#) pas_cbl_vdo
- struct [pd_do_t::ACT_CBL_VDO](#) act_cbl_vdo
- struct [pd_do_t::STD_AMA_VDO](#) std_ama_vdo
- struct [pd_do_t::STD_AMA_VDO_PD3](#) std_ama_vdo_pd3
- struct [pd_do_t::STD_SVID_RESP_VDO](#) std_svid_res
- struct [pd_do_t::STD_DP_VDO](#) std_dp_vdo
- struct [pd_do_t::DP_STATUS_VDO](#) dp_stat_vdo
- struct [pd_do_t::DP_CONFIG_VDO](#) dp_cfg_vdo

5.39.1 Detailed Description

Union to hold a PD data object. All USB-PD data objects are 4-byte values which are interpreted according to the message type, length and object position. This union represents all possible interpretations of a USB-PD data object.

5.39.2 Field Documentation

5.39.2.1 act_cbl_vdo

```
struct pd_do_t::ACT_CBL_VDO act_cbl_vdo
```

Active cable VDO structure as defined by PD 3.0.

5.39.2.2 bat_snk

```
struct pd_do_t::BAT_SNK bat_snk
```

DO interpreted as a Battery Supply PDO - Sink.

5.39.2.3 bat_src

```
struct pd_do_t::BAT_SRC bat_src
```

DO interpreted as a Battery Supply PDO - Source.

5.39.2.4 bist_do

```
struct pd_do_t::BIST_DO bist_do
```

DO interpreted as a BIST data object.

5.39.2.5 dp_cfg_vdo

```
struct pd_do_t::DP_CONFIG_VDO dp_cfg_vdo
```

DisplayPort configure VDO as defined by VESA spec.

5.39.2.6 dp_stat_vdo

```
struct pd_do_t::DP_STATUS_VDO dp_stat_vdo
```

DisplayPort status update VDO as defined by VESA spec.

5.39.2.7 fixed_snk

```
struct pd_do_t::FIXED_SNK fixed_snk
```

DO interpreted as a Fixed Supply PDO - Sink.

5.39.2.8 fixed_src

```
struct pd_do_t::FIXED_SRC fixed_src
```

DO interpreted as a Fixed Supply PDO - Source.

5.39.2.9 pas_cbl_vdo

```
struct pd_do_t::PAS_CBL_VDO pas_cbl_vdo
```

Passive cable VDO structure as defined by PD 3.0.

5.39.2.10 qc_4_0_data_vdo

```
struct pd_do_t::QC_4_0_DATA_VDO qc_4_0_data_vdo
```

Unstructured VDM data object as defined by QC 4.0 spec.

5.39.2.11 rdo_bat

```
struct pd_do_t::RDO_BAT rdo_bat
```

Battery Request Data Object.

5.39.2.12 rdo_bat_gvb

```
struct pd_do_t::RDO_BAT_GIVEBACK rdo_bat_gvb
```

Battery Request Data Object with GiveBack.

5.39.2.13 rdo_fix_var

```
struct pd_do_t::RDO_FIXED_VAR rdo_fix_var
```

Fixed or Variable Request Data Object.

5.39.2.14 rdo_fix_var_gvb

```
struct pd_do_t::RDO_FIXED_VAR_GIVEBACK rdo_fix_var_gvb
```

Fixed or Variable Request Data Object with GiveBack.

5.39.2.15 rdo_gen

```
struct pd_do_t::RDO_GEN rdo_gen
```

Generic Request Data Object representation.

5.39.2.16 rdo_gen_gvb

```
struct pd_do_t::RDO_GEN_GVB rdo_gen_gvb
```

Generic Request Data Object with GiveBack representation.

5.39.2.17 src_gen

```
struct pd_do_t::SRC_GEN src_gen
```

Generic source DO.

5.39.2.18 std_ama_vdo

```
struct pd_do_t::STD_AMA_VDO std_ama_vdo
```

AMA VDO structure as defined by PD 2.0.

5.39.2.19 std_ama_vdo_pd3

```
struct pd_do_t::STD_AMA_VDO_PD3 std_ama_vdo_pd3
```

AMA VDO structure as defined by PD 3.0.

5.39.2.20 std_cbl_vdo

```
struct pd_do_t::STD_CBL_VDO std_cbl_vdo
```

Cable VDO structure as defined in USB-PD r2.0.

5.39.2.21 std_cert_vdo

```
struct pd_do_t::STD_CERT_VDO std_cert_vdo
```

Cert Stat VDO structure.

5.39.2.22 std_dp_vdo

```
struct pd_do_t::STD_DP_VDO std_dp_vdo
```

DisplayPort Mode VDO as defined by VESA spec.

5.39.2.23 std_id_hdr

```
struct pd_do_t::STD_VDM_ID_HDR std_id_hdr
```

Standard ID_HEADER VDO.

5.39.2.24 std_prod_vdo

```
struct pd_do_t::STD_PROD_VDO std_prod_vdo
```

Product VDO structure.

5.39.2.25 std_svid_res

```
struct pd_do_t::STD_SVID_RESP_VDO std_svid_res
```

Discover_SVID response structure.

5.39.2.26 std_vdm_hdr

```
struct pd_do_t::STD_VDM_HDR std_vdm_hdr
```

Structured VDM header data object.

5.39.2.27 ustd_qc_4_0_hdr

```
struct pd_do_t::USTD_QC_4_0_HDR ustd_qc_4_0_hdr
```

Unstructured VDM header data object as defined by QC 4.0 spec.

5.39.2.28 ustd_vdm_hdr

```
struct pd_do_t::USTD_VDM_HDR ustd_vdm_hdr
```

Unstructured VDM header data object as defined by Cypress.

5.39.2.29 val

```
uint32_t val
```

Data object interpreted as an unsigned integer value.

5.39.2.30 var_snk

```
struct pd_do_t::VAR_SNK var_snk
```

DO interpreted as a Variable Supply PDO - Sink.

5.39.2.31 var_src

```
struct pd_do_t::VAR_SRC var_src
```

DO interpreted as a Variable Supply PDO - Source.

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

- pd_common/pd.h

5.40 pd_extd_hdr_t Union Reference

```
#include <pd.h>
```

Data Fields

- `uint16_t val`
- `struct {`
 - `uint16_t data_size: 9`
 - `uint16_t rsvd1: 1`
 - `uint16_t request: 1`
 - `uint16_t chunk_no: 4`
 - `uint16_t chunked: 1``} extd`

5.40.1 Detailed Description

Union to hold the PD extended header.

5.40.2 Field Documentation

5.40.2.1 chunk_no

```
uint16_t chunk_no
```

Bits 14:11 - Chunk number.

5.40.2.2 chunked

```
uint16_t chunked
```

Bit 15 - Chunked message.

5.40.2.3 data_size

```
uint16_t data_size
```

Bits 08:00 - Extended message size in bytes.

5.40.2.4 extd

```
struct { ... } extd
```

Extended header broken down into respective fields.

5.40.2.5 request

```
uint16_t request
```

Bit 10 - Chunk request.

5.40.2.6 rsvd1

`uint16_t rsvd1`

Bit 09 - Reserved.

5.40.2.7 val

`uint16_t val`

Extended header expressed as 2-byte integer value.

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

- [pd_common/pd.h](#)

5.41 pd_hdr_t::PD_HDR Struct Reference

Data Fields

- `uint32_t msg_type: 5`
- `uint32_t data_role: 1`
- `uint32_t spec_rev: 2`
- `uint32_t pwr_role: 1`
- `uint32_t msg_id: 3`
- `uint32_t len: 3`
- `uint32_t extd: 1`
- `uint32_t data_size: 9`
- `uint32_t rsvd1: 1`
- `uint32_t request: 1`
- `uint32_t chunk_no: 4`
- `uint32_t chunked: 1`

5.41.1 Field Documentation

5.41.1.1 chunk_no

`uint32_t chunk_no`

Bits 30:27 - Chunk number.

5.41.1.2 chunked

`uint32_t chunked`

Bit 31 - Chunked message.

5.41.1.3 data_role

`uint32_t data_role`

Bit 05 - Data role.

5.41.1.4 data_size

```
uint32_t data_size
```

Bits 24:16 - Extended message size in bytes.

5.41.1.5 extd

```
uint32_t extd
```

Bit 15 - Extended message.

5.41.1.6 len

```
uint32_t len
```

Bits 14:12 - Number of data objects.

5.41.1.7 msg_id

```
uint32_t msg_id
```

Bits 11:09 - Message ID.

5.41.1.8 msg_type

```
uint32_t msg_type
```

Bits 04:00 - Message type.

5.41.1.9 pwr_role

```
uint32_t pwr_role
```

Bit 08 - Power role.

5.41.1.10 request

```
uint32_t request
```

Bit 26 - Chunk request.

5.41.1.11 rsvd1

```
uint32_t rsvd1
```

Bit 25 - Reserved.

5.41.1.12 spec_rev

```
uint32_t spec_rev
```

Bits 07:06 - Spec revision.

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

- pd_common/[pd.h](#)

5.42 pd_hdr_t Union Reference

```
#include <pd.h>
```

Data Structures

- struct [PD_HDR](#)

Data Fields

- uint32_t [val](#)
- struct [pd_hdr_t::PD_HDR](#) [hdr](#)

5.42.1 Detailed Description

Union to hold the PD header defined by the USB-PD specification. Lower 16 bits hold the message header and the upper 16 bits hold the extended message header (where applicable).

5.42.2 Field Documentation

5.42.2.1 [hdr](#)

```
struct pd\_hdr\_t::PD\_HDR hdr
```

Header broken down into respective fields.

5.42.2.2 [val](#)

```
uint32_t val
```

Header expressed as a 32-bit word.

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

- [pd_common/pd.h](#)

5.43 pd_packet_extd_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t [sop](#)
- uint8_t [len](#)
- uint8_t [msg](#)
- uint8_t [data_role](#)
- [pd_hdr_t](#) [hdr](#)
- [pd_do_t](#) [dat](#) [MAX_EXTD_PKT_WORDS]

5.43.1 Detailed Description

Struct to hold extended PD packets (messages).

5.43.2 Field Documentation

5.43.2.1 dat

`pd_do_t` `dat [MAX_EXTD_PKT_WORDS]`

Data associated with the message.

5.43.2.2 data_role

`uint8_t` `data_role`

Data role.

5.43.2.3 hdr

`pd_hdr_t` `hdr`

Message header, including extended header.

5.43.2.4 len

`uint8_t` `len`

Length of the message: Unused for unchunked extended messages.

5.43.2.5 msg

`uint8_t` `msg`

Message code.

5.43.2.6 sop

`uint8_t` `sop`

Packet type.

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

- `pd_common/pd.h`

5.44 pd_packet_t Struct Reference

```
#include <pd.h>
```

Data Fields

- `uint8_t sop`
- `uint8_t len`
- `uint8_t msg`
- `uint8_t data_role`
- `pd_hdr_t hdr`
- `pd_do_t dat [MAX_NO_OF_DO]`

5.44.1 Detailed Description

Struct to hold a PD packet.

5.44.2 Field Documentation

5.44.2.1 dat

`pd_do_t dat [MAX_NO_OF_DO]`

Data objects associated with the message.

5.44.2.2 data_role

`uint8_t data_role`

Data role.

5.44.2.3 hdr

`pd_hdr_t hdr`

Message header.

5.44.2.4 len

`uint8_t len`

Length in data objects.

5.44.2.5 msg

`uint8_t msg`

Message code.

5.44.2.6 sop

`uint8_t sop`

Packet type.

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

- `pd_common/pd.h`

5.45 pd_port_config_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint16_t `id_vdm_offset`
- uint16_t `id_vdm_length`
- uint16_t `svid_vdm_offset`
- uint16_t `svid_vdm_length`
- uint16_t `mode_vdm_offset`
- uint16_t `mode_vdm_length`
- uint16_t `ext_src_cap_offset`
- uint16_t `ext_src_cap_length`
- uint16_t `reserved_0` [4]
- uint16_t `ra_timeout`
- uint16_t `reserved_1` [3]
- uint8_t `port_role`
- uint8_t `default_role`
- uint8_t `current_level`
- uint8_t `is_src_bat`
- uint8_t `is_snk_bat`
- uint8_t `snk_usb_susp_en`
- uint8_t `snk_usb_comm_en`
- uint8_t volatile `swap_response`
- uint8_t `drp_toggle_en`
- uint8_t `src_pdo_cnt`
- uint8_t `default_src_pdo_mask`
- uint8_t `snk_pdo_cnt`
- uint8_t `default_sink_pdo_mask`
- uint8_t `rp_supported`
- uint8_t `pd_operation_en`
- uint8_t `try_src_en`
- uint8_t `cable_disc_en`
- uint8_t `dead_bat_support`
- uint8_t `err_recovery_en`
- uint8_t volatile `port_disable`
- uint8_t volatile `frs_enable`
- uint8_t volatile `vconn_retain`
- uint16_t `reserved_3` [5]
- uint32_t `src_pdo_list` [`MAX_NO_OF_PDO`]
- uint32_t `snk_pdo_list` [`MAX_NO_OF_PDO`]
- uint16_t `snk_pdo_max_min_current_pwr` [`MAX_NO_OF_PDO`]
- uint16_t `reserved_4`
- uint8_t volatile `protection_enable`
- uint8_t `reserved_8`
- uint16_t `ovp_tbl_offset`
- uint16_t `ocp_tbl_offset`
- uint16_t `uvp_tbl_offset`
- uint16_t `scp_tbl_offset`
- uint16_t `vconn_ocp_tbl_offset`
- uint16_t `otp_tbl_offset`
- uint16_t `pwr_tbl_offset`
- uint16_t `chg_cfg_tbl_offset`

- uint16_t `bat_chg_tbl_offset`
- uint8_t `reserved_9` [4]
- uint8_t volatile `dp_config_supported`
- uint8_t volatile `dp_mux_control`
- uint8_t volatile `dp_mode_trigger`
- uint8_t `dp_oper`
- uint8_t volatile `dp_pref_mode`
- uint8_t `reserved_6` [3]
- uint16_t `bb_tbl_offset`
- uint8_t volatile `type_a_enable`
- uint8_t `reserved_10`
- uint16_t `type_a_pwr_tbl_offset`
- uint16_t `type_a_chg_tbl_offset`
- uint16_t `dock_cfg_tbl_offset`
- uint16_t `spm_cfg_tbl_offset`
- uint8_t `reserved_11` [60]

5.45.1 Detailed Description

PD port-specific configuration data from the configuration table.

5.45.2 Field Documentation

5.45.2.1 `bat_chg_tbl_offset`

`uint16_t bat_chg_tbl_offset`

Byte 0x9A: Offset to battery charging parameters.

5.45.2.2 `bb_tbl_offset`

`uint16_t bb_tbl_offset`

Byte 0xA8: Two byte offset to Billboard settings

5.45.2.3 `cable_disc_en`

`uint8_t cable_disc_en`

Byte 0x30: Whether cable discovery is supported on the port.

5.45.2.4 `chg_cfg_tbl_offset`

`uint16_t chg_cfg_tbl_offset`

Byte 0x98: Offset to legacy charging parameters.

5.45.2.5 `current_level`

`uint8_t current_level`

Byte 0x22: Type-C current level: 0=Default, 1=1.5A, 2=3A.

5.45.2.6 dead_bat_support

```
uint8_t dead_bat_support
```

Byte 0x31: Whether firmware should force Sink operation in Dead Battery condition.

5.45.2.7 default_role

```
uint8_t default_role
```

Byte 0x21: Default port role in case of a dual role port: 0=Sink, 1=Source.

5.45.2.8 default_sink_pdo_mask

```
uint8_t default_sink_pdo_mask
```

Byte 0x2C: Default Sink PDO enable mask.

5.45.2.9 default_src_pdo_mask

```
uint8_t default_src_pdo_mask
```

Byte 0x2A: Default Source PDO enable mask.

5.45.2.10 dock_cfg_tbl_offset

```
uint16_t dock_cfg_tbl_offset
```

Byte 0xB0: Offset to dock configuration parameter table.

5.45.2.11 dp_config_supported

```
uint8_t volatile dp_config_supported
```

Byte 0xA0: Supported Pin configurations for DP modes
0b00000000: USB SS only. 0b00000001: Reserved for future use (A).
0b00000010: Reserved for future use (B). 0b00000100: Pin Config C.
0b00001000: Pin Config D. 0b00100000: Pin Config E.
0b00100000: Pin Config F.

5.45.2.12 dp_mode_trigger

```
uint8_t volatile dp_mode_trigger
```

Byte 0xA2: DP_MODE_TRIGGER: Trigger for initializing DP modes. 0 => CCG initiates DP after contract. 1 => CCG waits for a trigger from EC.

5.45.2.13 dp_mux_control

```
uint8_t volatile dp_mux_control
```

Byte 0xA1: DP_MUX_CONTROL method: 0 => DP MUX Controlled by CCG. 1 => DP MUX Controlled by EC.

5.45.2.14 dp_oper

```
uint8_t dp_oper
```

Byte 0xA3: Type of DP operation supported. Bit 0: DP Sink supported Bit 1: DP Source supported.

5.45.2.15 dp_pref_mode

```
uint8_t volatile dp_pref_mode
```

Byte 0xA4: DP preferred mode. Bit 0: 0: CCG as DP Sink prefers 4 lane DP mode only. 1: CCG as DP Sink prefers 2 lane DP + USB SS Mode. All other bits are reserved.

5.45.2.16 drp_toggle_en

```
uint8_t drp_toggle_en
```

Byte 0x28: Whether DRP toggle is enabled.

5.45.2.17 err_recovery_en

```
uint8_t err_recovery_en
```

Byte 0x32: Whether PD error recovery is enabled.

5.45.2.18 ext_src_cap_length

```
uint16_t ext_src_cap_length
```

Byte 0x0E: Two byte length of the Src. Cap Extended response.

5.45.2.19 ext_src_cap_offset

```
uint16_t ext_src_cap_offset
```

Byte 0x0C: Two byte offset to the Src. Cap Extended response.

5.45.2.20 frs_enable

```
uint8_t volatile frs_enable
```

Byte 0x34: Fast Role Swap enable flags.

5.45.2.21 id_vdm_length

```
uint16_t id_vdm_length
```

Byte 0x02: Two byte length of the Discover ID Response VDM.

5.45.2.22 id_vdm_offset

```
uint16_t id_vdm_offset
```

Byte 0x00: Two byte offset to the Discover ID Response VDM.

5.45.2.23 is_snk_bat

```
uint8_t is_snk_bat
```

Byte 0x24: Whether the power sink is connected to a battery.

5.45.2.24 is_src_bat

```
uint8_t is_src_bat
```

Byte 0x23: Whether the power source is connected to a battery.

5.45.2.25 mode_vdm_length

```
uint16_t mode_vdm_length
```

Byte 0x0A: Two byte length of the Discover Mode Response VDM.

5.45.2.26 mode_vdm_offset

```
uint16_t mode_vdm_offset
```

Byte 0x08: Two byte offset to the Discover Mode Response VDM.

5.45.2.27 ocp_tbl_offset

```
uint16_t ocp_tbl_offset
```

Byte 0x8C: Offset to VBus OCP settings.

5.45.2.28 otp_tbl_offset

```
uint16_t otp_tbl_offset
```

Byte 0x94: Offset to OTP settings.

5.45.2.29 ovp_tbl_offset

```
uint16_t ovp_tbl_offset
```

Byte 0x8A: Offset to VBus OVP settings.

5.45.2.30 pd_operation_en

```
uint8_t pd_operation_en
```

Byte 0x2E: Whether PD operation is supported on the port.

5.45.2.31 port_disable

```
uint8_t volatile port_disable
```

Byte 0x33: Port disable flag.

5.45.2.32 port_role

```
uint8_t port_role
```

Byte 0x20: PD port role: 0=Sink, 1=Source, 2=Dual Role.

5.45.2.33 protection_enable

```
uint8_t volatile protection_enable
```

Byte 0x88: Enable field for protection settings Bit0: ovp enable Bit1: ocp enable Bit2: uvp enable Bit3: scp enable Bit4: vconn ocp enable Bit5: otp enable Bit(6:7): Reserved for future use

5.45.2.34 pwr_tbl_offset

```
uint16_t pwr_tbl_offset
```

Byte 0x96: Offset to power parameters.

5.45.2.35 ra_timeout

```
uint16_t ra_timeout
```

Byte 0x18: Ra removal delay for EMCA applications, measured in ms.

5.45.2.36 reserved_0

```
uint16_t reserved_0[4]
```

Byte 0x10: Reserved area for additional VDMs.

5.45.2.37 reserved_1

```
uint16_t reserved_1[3]
```

Byte 0x1A: Reserved area for EMCA configuration.

5.45.2.38 reserved_10

```
uint8_t reserved_10
```

Byte 0xAB: Reserved.

5.45.2.39 reserved_11

```
uint8_t reserved_11[60]
```

Byte 0xB4: Reserved area for future expansion.

5.45.2.40 reserved_3

```
uint16_t reserved_3[5]
```

Byte 0x36: Reserved words for padding to 4-byte aligned address.

5.45.2.41 reserved_4

```
uint16_t reserved_4
```

Byte 0x86: Reserved space for additional port parameters.

5.45.2.42 reserved_6

```
uint8_t reserved_6[3]
```

Byte 0xA5: Reserved area for future expansion.

5.45.2.43 reserved_8

```
uint8_t reserved_8
```

Byte 0x89: Reserved for future use.

5.45.2.44 reserved_9

```
uint8_t reserved_9[4]
```

Byte 0x9C: Reserved.

5.45.2.45 rp_supported

```
uint8_t rp_supported
```

Byte 0x2D: Supported Rp values. Multiple bits can be set. Bit 0 => Default current support. Bit 1 => 1.5A support. Bit 2 => 3A support.

5.45.2.46 scp_tbl_offset

```
uint16_t scp_tbl_offset
```

Byte 0x90: Offset to VBus SCP settings.

5.45.2.47 snk_pdo_cnt

```
uint8_t snk_pdo_cnt
```

Byte 0x2B: Number of valid sink PDOs in the table: Maximum supported value is 7.

5.45.2.48 snk_pdo_list

```
uint32_t snk_pdo_list [MAX_NO_OF_PDO]
```

Byte 0x5C: Sink PDO data array.

5.45.2.49 snk_pdo_max_min_current_pwr

```
uint16_t snk_pdo_max_min_current_pwr [MAX_NO_OF_PDO]
```

Byte 0x78: Array of sink PDO parameters. For each element, the format is as below: Bit 15 => Give Back support flag Bits 14:0 => Minimum sink operating current.

5.45.2.50 snk_usb_comm_en

```
uint8_t snk_usb_comm_en
```

Byte 0x26: Whether USB communication is supported.

5.45.2.51 snk_usb_susp_en

```
uint8_t snk_usb_susp_en
```

Byte 0x25: Whether USB suspend is supported.

5.45.2.52 spm_cfg_tbl_offset

```
uint16_t spm_cfg_tbl_offset
```

Byte 0xB2: Offset to Source Policy Manager parameter table.

5.45.2.53 src_pdo_cnt

```
uint8_t src_pdo_cnt
```

Byte 0x29: Number of valid source PDOs in the table: Maximum supported value is 7.

5.45.2.54 src_pdo_list

```
uint32_t src_pdo_list [MAX_NO_OF_PDO]
```

Byte 0x40: Source PDO data array.

5.45.2.55 svid_vdm_length

```
uint16_t svid_vdm_length
```

Byte 0x06: Two byte length of the Discover SVID Response VDM.

5.45.2.56 svid_vdm_offset

```
uint16_t svid_vdm_offset
```

Byte 0x04: Two byte offset to the Discover SVID Response VDM.

5.45.2.57 swap_response

```
uint8_t volatile swap_response
```

Byte 0x27: Response to be sent for each USB-PD SWAP command: Bits 1:0 => DR_SWAP response Bits 3:2 => PR_SWAP response Bits 5:4 => VCONN_SWAP response Allowed values are: 0=ACCEPT, 1=REJECT, 2=WAIT.

5.45.2.58 try_src_en

```
uint8_t try_src_en
```

Byte 0x2F: Whether Try.SRC state is supported on the port.

5.45.2.59 type_a_chg_tbl_offset

```
uint16_t type_a_chg_tbl_offset
```

Byte 0xAE: Offset to battery charging parameters of Type-A port.

5.45.2.60 type_a_enable

```
uint8_t volatile type_a_enable
```

Byte 0xAA: Type-A port support.

5.45.2.61 type_a_pwr_tbl_offset

```
uint16_t type_a_pwr_tbl_offset
```

Byte 0xAC: Offset to power parameters of Type-A port.

5.45.2.62 uvp_tbl_offset

```
uint16_t uvp_tbl_offset
```

Byte 0x8E: Offset to VBus UVP settings.

5.45.2.63 vconn_ocp_tbl_offset

```
uint16_t vconn_ocp_tbl_offset
```

Byte 0x92: Offset to Vcon OCP settings.

5.45.2.64 vconn_retain

```
uint8_t volatile vconn_retain
```

Byte 0x35: Whether VConn should be retained in ON state.

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

- [pd_common/pd.h](#)

5.46 pd_port_status_t::PD_PORT_STAT Struct Reference

Data Fields

- `uint32_t dflt_data_role: 2`
- `uint32_t dflt_data_pref: 1`
- `uint32_t dflt_power_role: 2`
- `uint32_t dflt_power_pref: 1`
- `uint32_t cur_data_role: 1`
- `uint32_t reserved0: 1`
- `uint32_t cur_power_role: 1`
- `uint32_t min_state: 1`
- `uint32_t contract_exist: 1`
- `uint32_t emca_present: 1`
- `uint32_t vconn_src: 1`
- `uint32_t vconn_on: 1`
- `uint32_t rp_status: 1`
- `uint32_t pe_rdy: 1`
- `uint32_t ccg_spec_rev: 2`
- `uint32_t peer_pd3_supp: 1`
- `uint32_t peer_unchunk_supp: 1`
- `uint32_t emca_spec_rev: 2`
- `uint32_t reserved2: 10`

5.46.1 Field Documentation

5.46.1.1 ccg_spec_rev

```
uint32_t ccg_spec_rev
```

USB-PD revision supported by CCG firmware.

5.46.1.2 contract_exist

```
uint32_t contract_exist
```

Whether explicit contract exists.

5.46.1.3 cur_data_role

```
uint32_t cur_data_role
```

Current data role.

5.46.1.4 cur_power_role

```
uint32_t cur_power_role
```

Current power role.

5.46.1.5 dfilt_data_pref

```
uint32_t dfilt_data_pref
```

Preferred data role in case of DRP.

5.46.1.6 dfilt_data_role

```
uint32_t dfilt_data_role
```

Default data role.

5.46.1.7 dfilt_power_pref

```
uint32_t dfilt_power_pref
```

Preferred power role in case of DRP.

5.46.1.8 dfilt_power_role

```
uint32_t dfilt_power_role
```

Default power role.

5.46.1.9 emca_present

```
uint32_t emca_present
```

EMCA detected or not.

5.46.1.10 emca_spec_rev

```
uint32_t emca_spec_rev
```

USB-PD revision supported by cable marker.

5.46.1.11 min_state

```
uint32_t min_state
```

Whether in Min state (due to GotoMin).

5.46.1.12 pe_rdy

```
uint32_t pe_rdy
```

Whether Policy Engine is in Ready state.

5.46.1.13 peer_pd3_supp

```
uint32_t peer_pd3_supp
```

Whether port partner supports PD 3.0.

5.46.1.14 peer_unchunk_supp

```
uint32_t peer_unchunk_supp
```

Whether port partner supports unchunked messages.

5.46.1.15 reserved0

```
uint32_t reserved0
```

Reserved.

5.46.1.16 reserved2

```
uint32_t reserved2
```

Reserved field.

5.46.1.17 rp_status

```
uint32_t rp_status
```

Current Rp status.

5.46.1.18 vconn_on

```
uint32_t vconn_on
```

Whether VConn is actually ON.

5.46.1.19 vconn_src

```
uint32_t vconn_src
```

Whether CCG is VConn source.

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

- pd_common/[pd.h](#)

5.47 pd_port_status_t Union Reference

```
#include <pd.h>
```

Data Structures

- struct [PD_PORT_STAT](#)

Data Fields

- uint32_t [val](#)
- struct [pd_port_status_t::PD_PORT_STAT](#) [status](#)

5.47.1 Detailed Description

PD port status as reported to Embedded Controller.

5.47.2 Field Documentation

5.47.2.1 [status](#)

```
struct pd\_port\_status\_t::PD\_PORT\_STAT status
```

Structure containing status bits.

5.47.2.2 [val](#)

```
uint32_t val
```

PD-Status register value.

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

- pd_common/[pd.h](#)

5.48 pd_power_status_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t intl_temperature
- uint8_t present_input
- uint8_t battery_input
- uint8_t event_flags
- uint8_t temp_status
- uint8_t dummy [3]

5.48.1 Detailed Description

PD port status corresponding to the Status Data Block (SSDB) See Table 6-39 of USB-PD R3 specification.

5.48.2 Field Documentation

5.48.2.1 battery_input

```
uint8_t battery_input
```

Reports the current battery status.

5.48.2.2 dummy

```
uint8_t dummy[3]
```

Reserved field used for 4 byte alignment.

5.48.2.3 event_flags

```
uint8_t event_flags
```

Event flags.

5.48.2.4 intl_temperature

```
uint8_t intl_temperature
```

Port's internal temperature. 0 if not supported.

5.48.2.5 present_input

```
uint8_t present_input
```

Reports current input power status.

5.48.2.6 temp_status

```
uint8_t temp_status
```

Temperature status.

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

- pd_common/[pd.h](#)

5.49 pwr_params_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t fb_type
- uint16_t vbus_min_volt
- uint16_t vbus_max_volt
- uint16_t vbus_dflt_volt
- uint32_t fb_ctrl_r1
- uint32_t fb_ctrl_r2

5.49.1 Detailed Description

Struct to hold the port power parameters.

5.49.2 Field Documentation

5.49.2.1 fb_ctrl_r1

```
uint32_t fb_ctrl_r1
```

Feedback control circuit: R1 resistance in Ohms

5.49.2.2 fb_ctrl_r2

```
uint32_t fb_ctrl_r2
```

Feedback control circuit: R2 resistance in Ohms

5.49.2.3 fb_type

```
uint8_t fb_type
```

Type of power feedback: Bit 0 -> No feedback Bit 1 -> PWM Bit 2 -> Direct feedback Bit 3 -> Opto-isolator based feedback

5.49.2.4 table_len

```
uint8_t table_len
```

Power parameters table length in bytes

5.49.2.5 vbus_dflt_volt

```
uint16_t vbus_dflt_volt
```

Default VBus supply voltage when feedback control is tri-stated.

5.49.2.6 vbus_max_volt

```
uint16_t vbus_max_volt
```

VBus maximum voltage in mV

5.49.2.7 vbus_min_volt

```
uint16_t vbus_min_volt
```

VBus minimum voltage in mV

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

- [pd_common/pd.h](#)

5.50 pd_do_t::QC_4_0_DATA_VDO Struct Reference

Data Fields

- `uint32_t data_0: 8`
- `uint32_t data_1: 8`
- `uint32_t data_2: 8`
- `uint32_t data_3: 8`

5.50.1 Field Documentation

5.50.1.1 data_0

```
uint32_t data_0
```

Command data #0.

5.50.1.2 data_1

```
uint32_t data_1
```

Command data #1.

5.50.1.3 data_2

uint32_t data_2

Command data #2.

5.50.1.4 data_3

uint32_t data_3

Command data #3.

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

- pd_common/pd.h

5.51 pd_do_t::RDO_BAT Struct Reference

Data Fields

- uint32_t max_op_power: 10
- uint32_t op_power: 10
- uint32_t rsrvd1: 3
- uint32_t unchunk_sup: 1
- uint32_t no_usb_suspend: 1
- uint32_t usb_comm_cap: 1
- uint32_t cap_mismatch: 1
- uint32_t give_back_flag: 1
- uint32_t obj_pos: 3
- uint32_t rsrvd2: 1

5.51.1 Field Documentation

5.51.1.1 cap_mismatch

uint32_t cap_mismatch

Capability mismatch.

5.51.1.2 give_back_flag

uint32_t give_back_flag

GiveBack flag = 0.

5.51.1.3 max_op_power

uint32_t max_op_power

Maximum operating power in 250mW units.

5.51.1.4 no_usb_suspend

uint32_t no_usb_suspend

No USB suspend.

5.51.1.5 obj_pos

uint32_t obj_pos

Object position.

5.51.1.6 op_power

uint32_t op_power

Operating power in 250mW units.

5.51.1.7 rsrvd1

uint32_t rsrvd1

Reserved field.

5.51.1.8 rsrvd2

uint32_t rsrvd2

Reserved field.

5.51.1.9 unchunk_sup

uint32_t unchunk_sup

Unchunked extended messages supported.

5.51.1.10 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

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

- pd_common/[pd.h](#)

5.52 pd_do_t::RDO_BAT_GIVEBACK Struct Reference

Data Fields

- uint32_t [min_op_power](#): 10
- uint32_t [op_power](#): 10
- uint32_t [rsrvd1](#): 3
- uint32_t [unchunk_sup](#): 1
- uint32_t [no_usb_suspend](#): 1

- `uint32_t usb_comm_cap`: 1
- `uint32_t cap_mismatch`: 1
- `uint32_t give_back_flag`: 1
- `uint32_t obj_pos`: 3
- `uint32_t rsrvd2`: 1

5.52.1 Field Documentation

5.52.1.1 `cap_mismatch`

`uint32_t cap_mismatch`

Capability mismatch.

5.52.1.2 `give_back_flag`

`uint32_t give_back_flag`

GiveBack flag = 1.

5.52.1.3 `min_op_power`

`uint32_t min_op_power`

Minimum operating power in 250mW units.

5.52.1.4 `no_usb_suspend`

`uint32_t no_usb_suspend`

No USB suspend.

5.52.1.5 `obj_pos`

`uint32_t obj_pos`

Object position.

5.52.1.6 `op_power`

`uint32_t op_power`

Operating power in 250mW units.

5.52.1.7 `rsrvd1`

`uint32_t rsrvd1`

Reserved field.

5.52.1.8 rsrvd2

uint32_t rsrvd2

Reserved field.

5.52.1.9 unchunk_sup

uint32_t unchunk_sup

Unchunked extended messages supported.

5.52.1.10 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

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

- pd_common/[pd.h](#)

5.53 pd_do_t::RDO_FIXED_VAR Struct Reference

Data Fields

- uint32_t [max_op_current](#): 10
- uint32_t [op_current](#): 10
- uint32_t [rsrvd1](#): 3
- uint32_t [unchunk_sup](#): 1
- uint32_t [no_usb_suspend](#): 1
- uint32_t [usb_comm_cap](#): 1
- uint32_t [cap_mismatch](#): 1
- uint32_t [give_back_flag](#): 1
- uint32_t [obj_pos](#): 3
- uint32_t [rsrvd2](#): 1

5.53.1 Field Documentation

5.53.1.1 cap_mismatch

uint32_t cap_mismatch

Capability mismatch.

5.53.1.2 give_back_flag

uint32_t give_back_flag

GiveBack flag = 0.

5.53.1.3 max_op_current

uint32_t max_op_current

Maximum operating current in 10mA units.

5.53.1.4 no_usb_suspend

uint32_t no_usb_suspend

No USB suspend.

5.53.1.5 obj_pos

uint32_t obj_pos

Object position.

5.53.1.6 op_current

uint32_t op_current

Operating current in 10mA units.

5.53.1.7 rsrvd1

uint32_t rsrvd1

Reserved field.

5.53.1.8 rsrvd2

uint32_t rsrvd2

Reserved field.

5.53.1.9 unchunk_sup

uint32_t unchunk_sup

Unchunked extended messages supported.

5.53.1.10 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

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

- pd_common/[pd.h](#)

5.54 pd_do_t::RDO_FIXED_VAR_GIVEBACK Struct Reference

Data Fields

- `uint32_t min_op_current: 10`
- `uint32_t op_current: 10`
- `uint32_t rsrvd1: 3`
- `uint32_t unchunk_sup: 1`
- `uint32_t no_usb_suspend: 1`
- `uint32_t usb_comm_cap: 1`
- `uint32_t cap_mismatch: 1`
- `uint32_t give_back_flag: 1`
- `uint32_t obj_pos: 3`
- `uint32_t rsrvd2: 1`

5.54.1 Field Documentation

5.54.1.1 `cap_mismatch`

`uint32_t cap_mismatch`

Capability mismatch.

5.54.1.2 `give_back_flag`

`uint32_t give_back_flag`

GiveBack flag = 1.

5.54.1.3 `min_op_current`

`uint32_t min_op_current`

Minimum operating current in 10mA units.

5.54.1.4 `no_usb_suspend`

`uint32_t no_usb_suspend`

No USB suspend.

5.54.1.5 `obj_pos`

`uint32_t obj_pos`

Object position.

5.54.1.6 `op_current`

`uint32_t op_current`

Operating current in 10mA units.

5.54.1.7 rsrvd1

uint32_t rsrvd1

Reserved field.

5.54.1.8 rsrvd2

uint32_t rsrvd2

Reserved field.

5.54.1.9 unchunk_sup

uint32_t unchunk_sup

Unchunked extended messages supported.

5.54.1.10 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

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

- pd_common/[pd.h](#)

5.55 pd_do_t::RDO_GEN Struct Reference

Data Fields

- uint32_t [min_max_power_cur](#): 10
- uint32_t [op_power_cur](#): 10
- uint32_t [rsrvd1](#): 3
- uint32_t [unchunk_sup](#): 1
- uint32_t [no_usb_suspend](#): 1
- uint32_t [usb_comm_cap](#): 1
- uint32_t [cap_mismatch](#): 1
- uint32_t [give_back_flag](#): 1
- uint32_t [obj_pos](#): 3
- uint32_t [rsrvd2](#): 1

5.55.1 Field Documentation

5.55.1.1 cap_mismatch

uint32_t cap_mismatch

Capability mismatch.

5.55.1.2 give_back_flag

uint32_t give_back_flag

GiveBack supported flag = 0.

5.55.1.3 min_max_power_cur

uint32_t min_max_power_cur

Min/Max power or current requirement.

5.55.1.4 no_usb_suspend

uint32_t no_usb_suspend

No USB suspend.

5.55.1.5 obj_pos

uint32_t obj_pos

Object position.

5.55.1.6 op_power_cur

uint32_t op_power_cur

Operating power or current requirement.

5.55.1.7 rsrvd1

uint32_t rsrvd1

Reserved field.

5.55.1.8 rsrvd2

uint32_t rsrvd2

Reserved field.

5.55.1.9 unchunk_sup

uint32_t unchunk_sup

Unchunked extended messages supported.

5.55.1.10 usb_comm_cap

uint32_t usb_comm_cap

USB communication capability.

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

- pd_common/[pd.h](#)

5.56 pd_do_t::RDO_GEN_GVB Struct Reference

Data Fields

- `uint32_t max_power_cur: 10`
- `uint32_t op_power_cur: 10`
- `uint32_t rsrvd1: 3`
- `uint32_t unchunk_sup: 1`
- `uint32_t no_usb_suspend: 1`
- `uint32_t usb_comm_cap: 1`
- `uint32_t cap_mismatch: 1`
- `uint32_t give_back_flag: 1`
- `uint32_t obj_pos: 3`
- `uint32_t rsrvd2: 1`

5.56.1 Field Documentation

5.56.1.1 cap_mismatch

`uint32_t cap_mismatch`

Capability mismatch.

5.56.1.2 give_back_flag

`uint32_t give_back_flag`

GiveBack supported flag = 1.

5.56.1.3 max_power_cur

`uint32_t max_power_cur`

Min/Max power or current requirement.

5.56.1.4 no_usb_suspend

`uint32_t no_usb_suspend`

No USB suspend.

5.56.1.5 obj_pos

`uint32_t obj_pos`

Object position.

5.56.1.6 op_power_cur

`uint32_t op_power_cur`

Operating power or current requirement.

5.56.1.7 rsrvd1

```
uint32_t rsrvd1
```

Reserved field.

5.56.1.8 rsrvd2

```
uint32_t rsrvd2
```

Reserved field.

5.56.1.9 unchunk_sup

```
uint32_t unchunk_sup
```

Unchunked extended messages supported.

5.56.1.10 usb_comm_cap

```
uint32_t usb_comm_cap
```

USB communication capability.

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

- pd_common/[pd.h](#)

5.57 ridge_reg_t Union Reference

```
#include <intel_ridge.h>
```

Data Structures

- struct [USBPD_CMD_REG](#)
- struct [USBPD_STATUS_REG](#)

Data Fields

- uint32_t [val](#)
- struct [ridge_reg_t::USBPD_STATUS_REG](#) [ridge_stat](#)
- struct [ridge_reg_t::USBPD_CMD_REG](#) [usbpd_cmd_reg](#)

5.57.1 Detailed Description

Over-Current status bit in the status register.

Union to hold AR/TR Registers

5.57.2 Field Documentation

5.57.2.1 ridge_stat

```
struct ridge_reg_t::USBPD_STATUS_REG ridge_stat
```

PD-controller status.

5.57.2.2 usbpd_cmd_reg

```
struct ridge_reg_t::USBPD_CMD_REG usbpd_cmd_reg
```

PD-controller command register.

5.57.2.3 val

```
uint32_t val
```

Integer field used for direct manipulation of reason code.

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

- app/alt_mode/intel_ridge.h

5.58 scp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t threshold
- uint8_t debounce
- uint8_t retry_cnt

5.58.1 Detailed Description

Struct to hold the SCP settings.

5.58.2 Field Documentation

5.58.2.1 debounce

```
uint8_t debounce
```

SCP debounce duration in micro-seconds. If a non-zero debounce is specified, there can be an error of upto 35 us due to the device being in sleep mode.

5.58.2.2 retry_cnt

```
uint8_t retry_cnt
```

Number of consecutive SCP events allowed before the port operation is suspended.

5.58.2.3 table_len

```
uint8_t table_len
```

Table length in bytes

5.58.2.4 threshold

```
uint8_t threshold
```

SCP threshold: Reduced voltage below expected value in percentage of expected voltage

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

- pd_common/[pd.h](#)

5.59 pd_do_t::SRC_GEN Struct Reference

Data Fields

- uint32_t [max_cur_power](#): 10
- uint32_t [min_voltage](#): 10
- uint32_t [max_voltage](#): 10
- uint32_t [supply_type](#): 2

5.59.1 Field Documentation

5.59.1.1 max_cur_power

```
uint32_t max_cur_power
```

Maximum current in 10 mA or power in 250 mW units.

5.59.1.2 max_voltage

```
uint32_t max_voltage
```

Maximum voltage in 50mV units.

5.59.1.3 min_voltage

```
uint32_t min_voltage
```

Minimum voltage in 50mV units.

5.59.1.4 supply_type

```
uint32_t supply_type
```

Supply type.

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

- pd_common/[pd.h](#)

5.60 pd_do_t::STD_AMA_VDO Struct Reference

Data Fields

- `uint32_t usb_ss_sup`: 3
- `uint32_t vbus_req`: 1
- `uint32_t vcon_req`: 1
- `uint32_t vcon_pwr`: 3
- `uint32_t ssrx2`: 1
- `uint32_t ssrx1`: 1
- `uint32_t sstx2`: 1
- `uint32_t sstx1`: 1
- `uint32_t rsvd1`: 12
- `uint32_t ama_fw_ver`: 4
- `uint32_t ama_hw_ver`: 4

5.60.1 Field Documentation

5.60.1.1 ama_fw_ver

`uint32_t ama_fw_ver`

AMA firmware version.

5.60.1.2 ama_hw_ver

`uint32_t ama_hw_ver`

AMA hardware version.

5.60.1.3 rsvd1

`uint32_t rsvd1`

Reserved field.

5.60.1.4 ssrx1

`uint32_t ssrx1`

Whether SSRX1 has configurable direction.

5.60.1.5 ssrx2

`uint32_t ssrx2`

Whether SSRX2 has configurable direction.

5.60.1.6 sstx1

`uint32_t sstx1`

Whether SSTX1 has configurable direction.

5.60.1.7 sstx2

uint32_t sstx2

Whether SSTX2 has configurable direction.

5.60.1.8 usb_ss_sup

uint32_t usb_ss_sup

USB signalling supported.

5.60.1.9 vbus_req

uint32_t vbus_req

Whether device requires VBus.

5.60.1.10 vcon_pwr

uint32_t vcon_pwr

VConn power required.

5.60.1.11 vcon_req

uint32_t vcon_req

Whether device requires VConn.

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

- pd_common/[pd.h](#)

5.61 pd_do_t::STD_AMA_VDO_PD3 Struct Reference

Data Fields

- uint32_t [usb_ss_sup](#): 3
- uint32_t [vbus_req](#): 1
- uint32_t [vcon_req](#): 1
- uint32_t [vcon_pwr](#): 3
- uint32_t [rvsd1](#): 13
- uint32_t [vdo_version](#): 3
- uint32_t [ama_fw_ver](#): 4
- uint32_t [ama_hw_ver](#): 4

5.61.1 Field Documentation

5.61.1.1 ama_fw_ver

uint32_t ama_fw_ver

AMA firmware version.

5.61.1.2 ama_hw_ver

`uint32_t ama_hw_ver`

AMA hardware version.

5.61.1.3 rsvd1

`uint32_t rsvd1`

Reserved field.

5.61.1.4 usb_ss_sup

`uint32_t usb_ss_sup`

USB signalling supported.

5.61.1.5 vbus_req

`uint32_t vbus_req`

Whether device requires VBus.

5.61.1.6 vcon_pwr

`uint32_t vcon_pwr`

VConn power required.

5.61.1.7 vcon_req

`uint32_t vcon_req`

Whether device requires VConn.

5.61.1.8 vdo_version

`uint32_t vdo_version`

VDO version.

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

- pd_common/[pd.h](#)

5.62 pd_do_t::STD_CBL_VDO Struct Reference

Data Fields

- `uint32_t usb_ss_sup: 3`
- `uint32_t sop_dp: 1`
- `uint32_t vbus_thru_cbl: 1`
- `uint32_t vbus_cur: 2`
- `uint32_t ssrx2: 1`

- uint32_t [ssrx1](#): 1
- uint32_t [sstx2](#): 1
- uint32_t [sstx1](#): 1
- uint32_t [cbl_term](#): 2
- uint32_t [cbl_latency](#): 4
- uint32_t [typec_plug](#): 1
- uint32_t [typec_abc](#): 2
- uint32_t [rsvd1](#): 4
- uint32_t [cbl_fw_ver](#): 4
- uint32_t [cbl_hw_ver](#): 4

5.62.1 Field Documentation

5.62.1.1 [cbl_fw_ver](#)

uint32_t [cbl_fw_ver](#)

Cable firmware version.

5.62.1.2 [cbl_hw_ver](#)

uint32_t [cbl_hw_ver](#)

Cable hardware version.

5.62.1.3 [cbl_latency](#)

uint32_t [cbl_latency](#)

Cable latency.

5.62.1.4 [cbl_term](#)

uint32_t [cbl_term](#)

Cable termination and VConn power requirement.

5.62.1.5 [rsvd1](#)

uint32_t [rsvd1](#)

Reserved field.

5.62.1.6 [sop_dp](#)

uint32_t [sop_dp](#)

Whether SOP" controller is present.

5.62.1.7 [ssrx1](#)

uint32_t [ssrx1](#)

Whether SSRX1 has configurable direction.

5.62.1.8 ssrx2

```
uint32_t ssrx2
```

Whether SSRX2 has configurable direction.

5.62.1.9 sstx1

```
uint32_t sstx1
```

Whether SSTX1 has configurable direction.

5.62.1.10 sstx2

```
uint32_t sstx2
```

Whether SSTX2 has configurable direction.

5.62.1.11 typec_abc

```
uint32_t typec_abc
```

Cable plug type.

5.62.1.12 typec_plug

```
uint32_t typec_plug
```

Whether cable has a plug: Should be 0.

5.62.1.13 usb_ss_sup

```
uint32_t usb_ss_sup
```

USB signalling supported by the cable.

5.62.1.14 vbus_cur

```
uint32_t vbus_cur
```

VBus current supported by the cable.

5.62.1.15 vbus_thru_cbl

```
uint32_t vbus_thru_cbl
```

Whether cable allows VBus power through.

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

- pd_common/[pd.h](#)

5.63 pd_do_t::STD_CERT_VDO Struct Reference

Data Fields

- `uint32_t usb_xid`: 32

5.63.1 Field Documentation

5.63.1.1 `usb_xid`

`uint32_t usb_xid`

32-bit XID value.

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

- `pd_common/pd.h`

5.64 pd_do_t::STD_DP_VDO Struct Reference

Data Fields

- `uint32_t port_cap`: 2
- `uint32_t signal`: 4
- `uint32_t recep`: 1
- `uint32_t usb2_0`: 1
- `uint32_t dfp_d_pin`: 8
- `uint32_t upf_d_pin`: 8
- `uint32_t rsvd`: 8

5.64.1 Field Documentation

5.64.1.1 `dfp_d_pin`

`uint32_t dfp_d_pin`

DFP_D pin assignments supported.

5.64.1.2 `port_cap`

`uint32_t port_cap`

Port capability.

5.64.1.3 `recep`

`uint32_t recep`

Whether Type-C connector is plug or receptacle.

5.64.1.4 rsvd

uint32_t rsvd

Reserved field.

5.64.1.5 signal

uint32_t signal

Signalling supported.

5.64.1.6 ufp_d_pin

uint32_t ufp_d_pin

UFP_D pin assignments supported.

5.64.1.7 usb2_0

uint32_t usb2_0

USB 2.0 signalling required.

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

- pd_common/[pd.h](#)

5.65 pd_do_t::STD_PROD_VDO Struct Reference

Data Fields

- uint32_t [bcd_dev](#): 16
- uint32_t [usb_pid](#): 16

5.65.1 Field Documentation

5.65.1.1 bcd_dev

uint32_t bcd_dev

16-bit bcdDevice value.

5.65.1.2 usb_pid

uint32_t usb_pid

16-bit product ID.

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

- pd_common/[pd.h](#)

5.66 pd_do_t::STD_SVID_RESP_VDO Struct Reference

Data Fields

- uint32_t [svid_n1](#): 16
- uint32_t [svid_n](#): 16

5.66.1 Field Documentation

5.66.1.1 [svid_n](#)

uint32_t [svid_n](#)

SVID #2

5.66.1.2 [svid_n1](#)

uint32_t [svid_n1](#)

SVID #1

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

- [pd_common/pd.h](#)

5.67 pd_do_t::STD_VDM_HDR Struct Reference

Data Fields

- uint32_t [cmd](#): 5
- uint32_t [rsvd1](#): 1
- uint32_t [cmd_type](#): 2
- uint32_t [obj_pos](#): 3
- uint32_t [rsvd2](#): 2
- uint32_t [st_ver](#): 2
- uint32_t [vdm_type](#): 1
- uint32_t [svid](#): 16

5.67.1 Field Documentation

5.67.1.1 [cmd](#)

uint32_t [cmd](#)

VDM command id.

5.67.1.2 cmd_type

uint32_t cmd_type

VDM command type.

5.67.1.3 obj_pos

uint32_t obj_pos

Object position.

5.67.1.4 rsvd1

uint32_t rsvd1

Reserved field.

5.67.1.5 rsvd2

uint32_t rsvd2

Reserved field.

5.67.1.6 st_ver

uint32_t st_ver

Structured VDM version.

5.67.1.7 svid

uint32_t svid

SVID associated with VDM.

5.67.1.8 vdm_type

uint32_t vdm_type

VDM type = Structured.

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

- pd_common/[pd.h](#)

5.68 pd_do_t::STD_VDM_ID_HDR Struct Reference

Data Fields

- uint32_t [usb_vid](#): 16
- uint32_t [rsvd1](#): 10
- uint32_t [mod_support](#): 1
- uint32_t [prod_type](#): 3
- uint32_t [usb_dev](#): 1
- uint32_t [usb_host](#): 1

5.68.1 Field Documentation

5.68.1.1 mod_support

uint32_t mod_support

Reserved field. Whether alternate modes are supported.

5.68.1.2 prod_type

uint32_t prod_type

Product type as UFP.

5.68.1.3 usb_dev

uint32_t usb_dev

USB device supported.

5.68.1.4 usb_host

uint32_t usb_host

USB host supported.

5.68.1.5 usb_vid

uint32_t usb_vid

16-bit vendor ID.

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

- pd_common/pd.h

5.69 sys_fw_metadata_t Struct Reference

```
#include <boot.h>
```

Data Fields

- uint8_t fw_checksum
- uint32_t fw_entry
- uint16_t boot_last_row
- uint8_t reserved1 [2]
- uint32_t fw_size
- uint8_t reserved2 [3]
- uint8_t active_boot_app
- uint8_t boot_app_ver_status
- uint16_t boot_app_version
- uint16_t boot_app_id

- `uint16_t metadata_valid`
- `uint32_t fw_version`
- `uint32_t boot_seq`

5.69.1 Detailed Description

CCGx Firmware metadata structure.

This structure defines the format of the firmware metadata that is stored on device flash. The boot-loader uses the metadata to identify the firmware validity, location, size, start address etc. The metadata for the two runtime firmware images (FW1 and FW2) are located at fixed addresses (for each CCGx part), allowing the boot-loader to precisely locate and validate the flash content during boot-up.

5.69.2 Field Documentation

5.69.2.1 active_boot_app

```
uint8_t active_boot_app
```

10: Creator specific field. Not used in this implementation.

5.69.2.2 boot_app_id

```
uint16_t boot_app_id
```

14: Creator specific field. Not used in this implementation.

5.69.2.3 boot_app_ver_status

```
uint8_t boot_app_ver_status
```

11: Creator specific field. Not used in this implementation.

5.69.2.4 boot_app_version

```
uint16_t boot_app_version
```

12: Creator specific field. Not used in this implementation.

5.69.2.5 boot_last_row

```
uint16_t boot_last_row
```

05: Last Flash row of Bootloader or previous firmware.

5.69.2.6 boot_seq

```
uint32_t boot_seq
```

1C: Boot sequence number field. Boot-loader will load the valid FW copy that has the higher sequence number associated with it.

5.69.2.7 fw_checksum

```
uint8_t fw_checksum
```

00: Single Byte FW Checksum.

5.69.2.8 fw_entry

```
uint32_t fw_entry
```

01: FW Entry Address

5.69.2.9 fw_size

```
uint32_t fw_size
```

09: Size of Firmware.

5.69.2.10 fw_version

```
uint32_t fw_version
```

18: Creator specific field. Not used in this implementation.

5.69.2.11 metadata_valid

```
uint16_t metadata_valid
```

16: Metadata Valid field. Valid if contains "CY".

5.69.2.12 reserved1

```
uint8_t reserved1[2]
```

07: Reserved.

5.69.2.13 reserved2

```
uint8_t reserved2[3]
```

0D: Reserved.

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

- [system/boot.h](#)

5.70 usb_config_t Struct Reference

```
#include <usb.h>
```

Data Fields

- bool [bus_power](#)
- [usb_event_cb_t event_cb](#)
- [usb_setup_cb_t get_dscr_cb](#)

- `usb_setup_cb_t class_rqt_cb`
- `usb_setup_cb_t vendor_rqt_cb`
- `usb_setup_cb_t fallback_rqt_cb`

5.70.1 Detailed Description

USB device mode configuration information.

The structure holds the USB device mode configuration parameters.

5.70.2 Field Documentation

5.70.2.1 bus_power

`bool bus_power`

Whether the device is bus powered or self powered.

5.70.2.2 class_rqt_cb

`usb_setup_cb_t class_rqt_cb`

Callback function to be invoked when a class request is received.

5.70.2.3 event_cb

`usb_event_cb_t event_cb`

Callback function to be invoked when any USB events happen.

5.70.2.4 fallback_rqt_cb

`usb_setup_cb_t fallback_rqt_cb`

Callback function to be invoked when a standard request unknown to the stack is received.

5.70.2.5 get_dscr_cb

`usb_setup_cb_t get_dscr_cb`

Callback function to be invoked when a GET_DESCRIPTOR request is received.

5.70.2.6 vendor_rqt_cb

`usb_setup_cb_t vendor_rqt_cb`

Callback function to be invoked when a vendor request is received.

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

- `usb/usb.h`

5.71 usb_ep_handle_t Struct Reference

```
#include <usb.h>
```

Data Fields

- bool `is_out`
- bool `toggle`
- bool `enabled`

5.71.1 Detailed Description

USB endpoint handle.

The structure holds information about the USB endpoint. The structure is for internal use and no additional instance of the structure is expected to be created.

5.71.2 Field Documentation

5.71.2.1 `enabled`

```
bool enabled
```

Whether the endpoint is enabled.

5.71.2.2 `is_out`

```
bool is_out
```

Direction of endpoint.

5.71.2.3 `toggle`

```
bool toggle
```

Active data toggle.

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

- `usb/usb.h`

5.72 usb_handle_t Struct Reference

```
#include <usb.h>
```

Data Fields

- `uint16_t dev_stat`
- `usb_state_t state`
- `usb_state_t prev_state`

- `uint32_t int_event`
- `usb_config_t cfg`
- `bool suspend_check`
- `uint32_t sof_data`
- `usb_setup_pkt_t setup_pkt`
- `usb_setup_cb_t ep0_xfer_cb`
- `usb_ep0_state_t ep0_state`
- `bool ep0_toggle`
- `uint8_t * ep0_buffer`
- `uint16_t ep0_length`
- `bool ep0_zlp_rqd`
- `bool ep0_last`
- `uint8_t active_cfg`
- `uint8_t active_alt_inf [USB_NUM_INTERFACE]`
- `usb_ep_handle_t ep_handle [USB_NUM_EP]`

5.72.1 Detailed Description

USB device controller handle.

The structure holds information about the USB device controller. The structure is for internal use and no additional instance of the structure is expected to be created.

5.72.2 Field Documentation

5.72.2.1 active_alt_inf

`uint8_t active_alt_inf[USB_NUM_INTERFACE]`

Current alternate interface selected by SET_INTERFACE.

5.72.2.2 active_cfg

`uint8_t active_cfg`

Current configuration value set by SET_CONFIG.

5.72.2.3 cfg

`usb_config_t cfg`

USB configuration information.

5.72.2.4 dev_stat

`uint16_t dev_stat`

Device status to be returned during GET_STATUS call.

5.72.2.5 ep0_buffer

`uint8_t* ep0_buffer`

Current EP0 data transfer buffer.

5.72.2.6 ep0_last

bool ep0_last

Wether the current EP0 read / write request is the last.

5.72.2.7 ep0_length

uint16_t ep0_length

Pending transfer size for EP0.

5.72.2.8 ep0_state

[usb_ep0_state_t](#) ep0_state

Current EP0 transfer state.

5.72.2.9 ep0_toggle

bool ep0_toggle

Current EP0 toggle state.

5.72.2.10 ep0_xfer_cb

[usb_setup_cb_t](#) ep0_xfer_cb

Transfer completion callback.

5.72.2.11 ep0_zlp_rqd

bool ep0_zlp_rqd

Whether an ZLP need to be sent on EP0 to indicate a short transfer.

5.72.2.12 ep_handle

[usb_ep_handle_t](#) ep_handle[[USB_NUM_EP](#)]

Endpoint handle for internal state machine.

5.72.2.13 int_event

uint32_t int_event

Internal interrupt event state.

5.72.2.14 prev_state

[usb_state_t](#) prev_state

Previous USB state.

5.72.2.15 setup_pkt

```
usb_setup_pkt_t setup_pkt
```

Current USB setup packet.

5.72.2.16 sof_data

```
uint32_t sof_data
```

Internal SOF check data to identify a USB bus suspend.

5.72.2.17 state

```
usb_state_t state
```

Current USB state.

5.72.2.18 suspend_check

```
bool suspend_check
```

Internal flag indicating whether to check for suspend.

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

- [usb/usb.h](#)

5.73 usb_i2cm_t Struct Reference

```
#include <usb_i2cm.h>
```

Data Fields

- [usb_i2cm_state_t state](#)
- bool [preamble_state](#)
- uint8_t [preamble](#)
- uint8_t [ctrl](#)
- uint8_t [status](#)
- uint16_t [count](#)
- uint16_t [length](#)

5.73.1 Detailed Description

Internal USB-I2CM module handle structure.

No explicit structure for the handle is expected to be created outside of the USB-I2CM module implementation.

5.73.2 Field Documentation

5.73.2.1 count

```
uint16_t count
```

Pending count of the current transaction.

5.73.2.2 ctrl

```
uint8_t ctrl
```

Control flags for the current transaction.

5.73.2.3 length

```
uint16_t length
```

Length of the current transaction.

5.73.2.4 preamble

```
uint8_t preamble
```

Preamble to be used for the current transaction.

5.73.2.5 preamble_state

```
bool preamble_state
```

Indicates whether the current request is in preamble state.

5.73.2.6 state

```
usb_i2cm_state_t state
```

State of the bridge interface.

5.73.2.7 status

```
uint8_t status
```

Status of the current transaction.

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

- app/[usb_i2cm.h](#)

5.74 usb_setup_pkt_t Struct Reference

```
#include <usbconst.h>
```

Data Fields

- uint8_t attrib
- uint8_t cmd
- uint16_t value

- `uint16_t index`
- `uint16_t length`

5.74.1 Detailed Description

USB setup packet format.

The structure breaks the setup packet into various fields defined in the USB specification.

5.74.2 Field Documentation

5.74.2.1 attrib

`uint8_t attrib`

Request attributes: Bit7: 0=Host to device, 1=Device to host Bits6:5 0=Standard, 1=Class, 2=Vendor, 3=Reserved Bits4:0 0=Device, 1=Interface, 2=Endpoint, 3=Other, 4-31=Reserved

5.74.2.2 cmd

`uint8_t cmd`

Request command

5.74.2.3 index

`uint16_t index`

Index parameter for command

5.74.2.4 length

`uint16_t length`

Number of bytes to be transferred in data phase

5.74.2.5 value

`uint16_t value`

Value parameter for command

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

- `usb/usbconst.h`

5.75 USBARB16_REGS_T Struct Reference

```
#include <usb.h>
```

Data Fields

- volatile uint32_t **arb_rwx_wa16**
- volatile uint32_t **rsrvd0**
- volatile uint32_t **arb_rwx_ra16**
- volatile uint32_t **rsrvd1**
- volatile uint32_t **arb_rwx_dr16**
- volatile uint32_t **rsrvd2** [0x0B]

5.75.1 Detailed Description

The following are register definitions made for ease of use. Refer to the register definition macros for mode details.

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

- [usb/usb.h](#)

5.76 USBARB_REGS_T Struct Reference

```
#include <usb.h>
```

Data Fields

- volatile uint32_t **arb_epx_cfg**
- volatile uint32_t **arb_epx_int_en**
- volatile uint32_t **arb_epx_sr**
- volatile uint32_t **rsrvd0**
- volatile uint32_t **arb_rwx_wa**
- volatile uint32_t **arb_rwx_wa_msb**
- volatile uint32_t **arb_rwx_ra**
- volatile uint32_t **arb_rwx_ra_msb**
- volatile uint32_t **arb_rwx_dr**
- volatile uint32_t **rsrvd1** [8]

5.76.1 Detailed Description

The following are register definitions made for ease of use. Refer to the register definition macros for mode details.

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

- [usb/usb.h](#)

5.77 ridge_reg_t::USBPD_CMD_REG Struct Reference

```
#include <intel_ridge.h>
```

Data Fields

- `uint32_t tbt_host_conn: 1`
- `uint32_t soft_rst: 1`
- `uint32_t i2c_int_ack: 1`
- `uint32_t rsvd2: 1`
- `uint32_t usb_host_conn: 1`
- `uint32_t dp_host_conn: 1`
- `uint32_t rsvd3: 7`
- `uint32_t irq_ack: 1`
- `uint32_t hpd_irq: 1`
- `uint32_t hpd_lvl: 1`
- `uint32_t rsvd4: 16`

5.77.1 Detailed Description

Struct containing USB-PD controller command register fields for Alpine/Titan Ridge.

5.77.2 Field Documentation

5.77.2.1 `dp_host_conn`

`uint32_t dp_host_conn`

B5: Indicates that DP Host is connected (TBT device only).

5.77.2.2 `hpd_irq`

`uint32_t hpd_irq`

B14: HPD IRQ when acting as DP Sink.

5.77.2.3 `hpd_lvl`

`uint32_t hpd_lvl`

B15: HPD level when acting as DP Sink.

5.77.2.4 `i2c_int_ack`

`uint32_t i2c_int_ack`

B2: Alpine/Titan Ridge acknowledge for the interrupt.

5.77.2.5 `irq_ack`

`uint32_t irq_ack`

B13: IRQ ACK PD Controller to Titan Ridge.

5.77.2.6 rsvd2

```
uint32_t rsvd2
```

B3: Reserved.

5.77.2.7 rsvd3

```
uint32_t rsvd3
```

B[12-6]: Reserved.

5.77.2.8 rsvd4

```
uint32_t rsvd4
```

B[31-16]: Reserved.

5.77.2.9 soft_rst

```
uint32_t soft_rst
```

B1: Issue USB-PD Controller soft reset.

5.77.2.10 tbt_host_conn

```
uint32_t tbt_host_conn
```

B0: TBT host connected.

5.77.2.11 usb_host_conn

```
uint32_t usb_host_conn
```

B4: Indicates that USB Host is connected (TBT device only).

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

- app/alt_mode/intel_ridge.h

5.78 ridge_reg_t::USBPD_STATUS_REG Struct Reference

```
#include <intel_ridge.h>
```

Data Fields

- uint32_t [data_conn_pres](#): 1
- uint32_t [conn_orien](#): 1
- uint32_t [active_cbl](#): 1
- uint32_t [ovc_indn](#): 1
- uint32_t [usb2_conn](#): 1
- uint32_t [usb3_conn](#): 1
- uint32_t [usb3_speed](#): 1
- uint32_t [usb_dr](#): 1
- uint32_t [dp_conn](#): 1

- `uint32_t dp_role: 1`
- `uint32_t dp_pin_assign: 2`
- `uint32_t dbg_acc_mode: 1`
- `uint32_t irq_ack: 1`
- `uint32_t hpd_irq: 1`
- `uint32_t hpd_lvl: 1`
- `uint32_t tbt_conn: 1`
- `uint32_t tbt_type: 1`
- `uint32_t cbl_type: 1`
- `uint32_t pro_dock_detect: 1`
- `uint32_t act_link_train: 1`
- `uint32_t dbg_alt_m_conn: 1`
- `uint32_t rsvd: 1`
- `uint32_t force_lsx: 1`
- `uint32_t pwr: 1`
- `uint32_t tbt_cbl_spd: 3`
- `uint32_t tbt_cbl_gen: 2`
- `uint32_t rsrvd4: 1`
- `uint32_t interrupt_ack: 1`

5.78.1 Detailed Description

Struct containing USB-PD controller status to be reported to Alpine/Titan Ridge. Using the structure definition corresponding to Titan Ridge in all cases.

5.78.2 Field Documentation

5.78.2.1 `act_link_train`

```
uint32_t act_link_train
```

B20: Active TBT link training. From B23 of Cable MODE Response.

5.78.2.2 `active_cbl`

```
uint32_t active_cbl
```

B2: Active cable. From B22 of Cable MODE Response.

5.78.2.3 `cbl_type`

```
uint32_t cbl_type
```

B18: TBT cable type. From B21 of Cable MODE Response.

5.78.2.4 `conn_orien`

```
uint32_t conn_orien
```

B1: CC polarity.

5.78.2.5 data_conn_pres

```
uint32_t data_conn_pres
```

B0: Whether data connection is present.

5.78.2.6 dbg_acc_mode

```
uint32_t dbg_acc_mode
```

B12: USB Type-C Debug accessory mode.

5.78.2.7 dbg_alt_m_conn

```
uint32_t dbg_alt_m_conn
```

B21: NIDnT Alt mode defined in MIPI SVID = 0xFF03.

5.78.2.8 dp_conn

```
uint32_t dp_conn
```

B8: DP connection status.

5.78.2.9 dp_pin_assign

```
uint32_t dp_pin_assign
```

B[11-10]: DP pin assignment. 4-lane='b00 2-lane='b01

5.78.2.10 dp_role

```
uint32_t dp_role
```

B9: DP direction. Source=0, Sink=1.

5.78.2.11 force_lsx

```
uint32_t force_lsx
```

B23: Set to zero.

5.78.2.12 hpd_irq

```
uint32_t hpd_irq
```

B14: HPD IRQ received from DP Sink.

5.78.2.13 hpd_lvl

```
uint32_t hpd_lvl
```

B15: HPD level received from DP Sink.

5.78.2.14 interrupt_ack

```
uint32_t interrupt_ack
```

B31: Interrupt indication (Set by EC when in I2C slave Mode).

5.78.2.15 irq_ack

```
uint32_t irq_ack
```

B13: Set after receiving GoodCRC from Attention message with IRQ_HPD.

5.78.2.16 ovc_indn

```
uint32_t ovc_indn
```

B3: Over-Current indication.

5.78.2.17 pro_dock_detect

```
uint32_t pro_dock_detect
```

B19: Reporting of vPro Support.

5.78.2.18 pwr

```
uint32_t pwr
```

B24: Indicates PWR mismatch (Used only in TBT BPD).

5.78.2.19 rsvd

```
uint32_t rsvd
```

B22: Reserved.

5.78.2.20 rsvd4

```
uint32_t rsvd4
```

B30: Reserved.

5.78.2.21 tbt_cbl_gen

```
uint32_t tbt_cbl_gen
```

B[29-28]: Cable generation. From B20-19 of Cable MODE Response.

5.78.2.22 tbt_cbl_spd

```
uint32_t tbt_cbl_spd
```

B[27-25]: Cable speed. From B18-B16 of Cable MODE Response.

5.78.2.23 tbt_conn

```
uint32_t tbt_conn
```

B16: TBT connection status.

5.78.2.24 tbt_type

```
uint32_t tbt_type
```

B17: TBT type. From B16 of UFP MODE Response.

5.78.2.25 usb2_conn

```
uint32_t usb2_conn
```

B4: USB 2.0 connection. Set when no ALT MODES are active.

5.78.2.26 usb3_conn

```
uint32_t usb3_conn
```

B5: USB 3.1 connection. Set when no ALT MODES are active.

5.78.2.27 usb3_speed

```
uint32_t usb3_speed
```

B6: USB Gen2/Gen1 speed. From B18-B16 of Cable MODE Response.

5.78.2.28 usb_dr

```
uint32_t usb_dr
```

B7: Data role. DFP=0, UFP=1.

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

- app/alt_mode/intel_ridge.h

5.79 USBSIE_REGS_T Struct Reference

```
#include <usb.h>
```

Data Fields

- volatile uint32_t **sie_epx_cnt0**
- volatile uint32_t **sie_epx_cnt1**
- volatile uint32_t **sie_epx_cr0**
- volatile uint32_t **rsrvd0** [0x0D]

5.79.1 Detailed Description

The following are register definitions made for ease of use. Refer to the register definition macros for mode details.
The documentation for this struct was generated from the following file:

- [usb/usb.h](#)

5.80 pd_do_t::USTD_QC_4_0_HDR Struct Reference

Data Fields

- `uint32_t cmd_0`: 8
- `uint32_t cmd_1`: 7
- `uint32_t vdm_type`: 1
- `uint32_t svid`: 16

5.80.1 Field Documentation

5.80.1.1 cmd_0

`uint32_t cmd_0`

Command code #0.

5.80.1.2 cmd_1

`uint32_t cmd_1`

Command code #1.

5.80.1.3 svid

`uint32_t svid`

SVID associated with message.

5.80.1.4 vdm_type

`uint32_t vdm_type`

VDM type = Unstructured.

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

- [pd_common/pd.h](#)

5.81 pd_do_t::USTD_VDM_HDR Struct Reference

Data Fields

- `uint32_t cmd`: 5

- uint32_t **seq_num**: 3
- uint32_t **rsvd1**: 3
- uint32_t **cmd_type**: 2
- uint32_t **vdm_ver**: 2
- uint32_t **vdm_type**: 1
- uint32_t **svid**: 16

5.81.1 Field Documentation

5.81.1.1 cmd

uint32_t cmd

Command id.

5.81.1.2 cmd_type

uint32_t cmd_type

Command type.

5.81.1.3 rsvd1

uint32_t rsvd1

Reserved field.

5.81.1.4 seq_num

uint32_t seq_num

Sequence number.

5.81.1.5 svid

uint32_t svid

SVID associated with VDM.

5.81.1.6 vdm_type

uint32_t vdm_type

VDM type = Unstructured.

5.81.1.7 vdm_ver

uint32_t vdm_ver

VDM version.

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

- pd_common/[pd.h](#)

5.82 uvp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t threshold
- uint8_t debounce
- uint8_t retry_cnt

5.82.1 Detailed Description

Struct to hold the UVP settings.

5.82.2 Field Documentation

5.82.2.1 debounce

```
uint8_t debounce
```

UVP debounce duration in micro-seconds. If a non-zero debounce is specified, there can be an error of upto 35 us due to the device being in sleep mode.

5.82.2.2 retry_cnt

```
uint8_t retry_cnt
```

Number of consecutive UVP events allowed before the port operation is suspended by CCG firmware.

5.82.2.3 table_len

```
uint8_t table_len
```

Table length in bytes

5.82.2.4 threshold

```
uint8_t threshold
```

UVP threshold: Reduced voltage below expected value in percentage of expected voltage

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

- pd_common/pd.h

5.83 pd_do_t::VAR_SNK Struct Reference

Data Fields

- uint32_t op_current: 10

- `uint32_t min_voltage: 10`
- `uint32_t max_voltage: 10`
- `uint32_t supply_type: 2`

5.83.1 Field Documentation

5.83.1.1 max_voltage

`uint32_t max_voltage`

Maximum voltage in 50mV units.

5.83.1.2 min_voltage

`uint32_t min_voltage`

Minimum voltage in 50mV units.

5.83.1.3 op_current

`uint32_t op_current`

Operational current in 10mA units.

5.83.1.4 supply_type

`uint32_t supply_type`

Supply type - should be 'b10.

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

- `pd_common/pd.h`

5.84 pd_do_t::VAR_SRC Struct Reference

Data Fields

- `uint32_t max_current: 10`
- `uint32_t min_voltage: 10`
- `uint32_t max_voltage: 10`
- `uint32_t supply_type: 2`

5.84.1 Field Documentation

5.84.1.1 max_current

`uint32_t max_current`

Maximum current in 10mA units.

5.84.1.2 max_voltage

```
uint32_t max_voltage
```

Maximum voltage in 50mV units.

5.84.1.3 min_voltage

```
uint32_t min_voltage
```

Minimum voltage in 50mV units.

5.84.1.4 supply_type

```
uint32_t supply_type
```

Supply type - should be 'b10.

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

- pd_common/pd.h

5.85 vconn_ocp_settings_t Struct Reference

```
#include <pd.h>
```

Data Fields

- uint8_t table_len
- uint8_t threshold
- uint8_t debounce
- uint8_t reserved_0

5.85.1 Detailed Description

Struct to hold the Vconn OCP settings.

5.85.2 Field Documentation

5.85.2.1 debounce

```
uint8_t debounce
```

Vconn OCP debounce period in ms.

5.85.2.2 reserved_0

```
uint8_t reserved_0
```

Reserved for future use

5.85.2.3 table_len

```
uint8_t table_len
```

Table length in bytes

5.85.2.4 threshold

```
uint8_t threshold
```

Max. Vconn current allowed in 10 mA units.

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

- pd_common/pd.h

5.86 vdm_msg_info_t Struct Reference

```
#include <vdm_task_mngr.h>
```

Data Fields

- [pd_do_t vdm_header](#)
- [uint8_t sop_type](#)
- [uint8_t vdo_numb](#)
- [pd_do_t vdo \[MAX_NO_OF_VDO\]](#)

5.86.1 Detailed Description

Struct holds received/sent VDM info.

This struct holds received/sent VDM information which uses by VDM alternative modes managers.

5.86.2 Field Documentation

5.86.2.1 sop_type

```
uint8_t sop_type
```

VDM SOP type.

5.86.2.2 vdm_header

```
pd_do_t vdm_header
```

Holds VDM buffer.

5.86.2.3 vdo

```
pd_do_t vdo [MAX_NO_OF_VDO]
```

VDO objects buffer.

5.86.2.4 vdo_numb

```
uint8_t vdo_numb
```

Number of received VDOs in VDM.

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

- app/alt_mode/vdm_task_mngr.h

5.87 vdm_resp_t Struct Reference

```
#include <pd.h>
```

Data Fields

- `pd_do_t resp_buf [MAX_NO_OF_DO]`
- `uint8_t do_count`
- `vdm_ams_t no_resp`

5.87.1 Detailed Description

Struct to hold response to policy manager.

5.87.2 Field Documentation

5.87.2.1 do_count

```
uint8_t do_count
```

Data objects count

5.87.2.2 no_resp

```
vdm_ams_t no_resp
```

Response type.

5.87.2.3 resp_buf

```
pd_do_t resp_buf [MAX_NO_OF_DO]
```

Data objects buffer

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

- pd_common/pd.h

Chapter 6

File Documentation

6.1 app/alt_mode/alt_mode_hw.h File Reference

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

Data Structures

- union `alt_mode_hw_evt_t`
- struct `alt_mode_hw_evt_t::ALT_MODE_HW_EVT`

Macros

- #define `NO_DATA` (0u)
- #define `HPD_ENABLE_CMD` (0u)
- #define `HPD_DISABLE_CMD` (5u)

Typedefs

- typedef void(* `alt_mode_hw_cmd_cbk_t`) (uint8_t port, uint32_t command)

Enumerations

- enum `alt_mode_hw_t` { `ALT_MODE_MUX` = 1, `ALT_MODE_HPD` }
- enum `mux_select_t` {
 `MUX_CONFIG_ISOLATE`, `MUX_CONFIG_2_0`, `MUX_CONFIG_SS_ONLY`, `MUX_CONFIG_DP_2_LANE`,
 `MUX_CONFIG_DP_4_LANE`, `MUX_CONFIG RIDGE CUSTOM`, `MUX_CONFIG_INIT`, `MUX_CONFIG_deinit`
}
- enum `mux_poll_status_t` { `MUX_STATE_IDLE`, `MUX_STATE_FAIL`, `MUX_STATE_BUSY`, `MUX_STATE_SUCCESS` }

Functions

- void `alt_mode_hw_set_cbk` (uint8_t port, `alt_mode_hw_cmd_cbk_t` cbk)
- void `alt_mode_hw_deinit` (uint8_t port)
- bool `eval_app_alt_hw_cmd` (uint8_t port, uint8_t *cmd_param)

- bool `eval_hpd_cmd` (uint8_t port, uint32_t cmd)
- bool `eval_mux_cmd` (uint8_t port, uint32_t cmd)
- bool `set_mux` (uint8_t port, `mux_select_t` cfg, uint32_t custom_data)
- `mux_select_t get_mux_state` (uint8_t port)
- bool `alt_mode_hw_is_idle` (uint8_t port)
- void `alt_mode_hw_sleep` (uint8_t port)
- void `alt_mode_hw_wakeup` (uint8_t port)
- bool `dp_snk_get_hpd_state` (uint8_t port)

6.1.1 Detailed Description

Hardware control for Alternate mode implementation.

6.1.2 Enumeration Type Documentation

6.1.2.1 alt_mode_hw_t

enum `alt_mode_hw_t`

Application HW type values which are used in `alt_mode_hw_evt_t` structure.

Enumerator

<code>ALT_MODE_MUX</code>	HW type - MUX.
<code>ALT_MODE_HPD</code>	HW type - HPD transceiver/receiver.

6.1.2.2 mux_poll_status_t

enum `mux_poll_status_t`

@ typedef `mux_status_t` @ brief Possible states for the MUX handler.

Enumerator

<code>MUX_STATE_IDLE</code>	MUX idle state.
<code>MUX_STATE_FAIL</code>	MUX switch failed.
<code>MUX_STATE_BUSY</code>	MUX is busy.
<code>MUX_STATE_SUCCESS</code>	MUX switched successfully.

6.1.2.3 mux_select_t

enum `mux_select_t`

@ typedef `mux_select_t` @ brief Possible settings for the Type-C Data MUX. @ note This type should be extended to cover all possible modes for the MUX.

Enumerator

MUX_CONFIG_ISOLATE	Isolate configuration.
MUX_CONFIG_2_0	USB 2.0 configuration.
MUX_CONFIG_SS_ONLY	USB SS configuration.
MUX_CONFIG_DP_2_LANE	Two lane DP configuration.
MUX_CONFIG_DP_4_LANE	Four lane DP configuration.
MUX_CONFIG RIDGE CUSTOM	Alpine/Titan Ridge custom configuration.
MUX_CONFIG_INIT	Enables MUX functionality.
MUX_CONFIG_deinit	Disables MUX functionality.

6.1.3 Function Documentation

6.1.3.1 alt_mode_hw_deinit()

```
void alt_mode_hw_deinit (
    uint8_t port )
```

This function deinit all HW related to alt modes.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

None.

6.1.3.2 alt_mode_hw_is_idle()

```
bool alt_mode_hw_is_idle (
    uint8_t port )
```

Check whether the ALT. MODE hardware block is idle.

This function is part of the deep-sleep entry checks for the CCG device, and checks whether there are any pending ALT. MODE hardware commands that require the device to be active.

Parameters

<i>port</i>	Port on which the hardware status is to be checked.
-------------	---

Returns

true if the block is idle, false otherwise.

6.1.3.3 alt_mode_hw_set_cbk()

```
void alt_mode_hw_set_cbk (
    uint8_t port,
    alt_mode_hw_cmd_cbk_t cbk )
```

This function registers an ALT. MODE hardware callback function.

Parameters

<i>port</i>	Port for which the callback is registered.
<i>cbk</i>	Callback function for ALT. MODE hardware events.

Returns

None.

6.1.3.4 alt_mode_hw_sleep()

```
void alt_mode_hw_sleep (
    uint8_t port )
```

Prepare the Alt. Mode hardware state for device deep-sleep.

Parameters

<i>port</i>	The port whose hardware state is to be configured.
-------------	--

Returns

None.

6.1.3.5 alt_mode_hw_wakeup()

```
void alt_mode_hw_wakeup (
    uint8_t port )
```

Restore Alt. Mode hardware state after device resumes from deep-sleep.

Parameters

<i>port</i>	The port whose hardware state is to be restored.
-------------	--

Returns

None

6.1.3.6 dp_snk_get_hpdu_state()

```
bool dp_snk_get_hpdu_state (
    uint8_t port )
```

Return HPD state based on HPD Queue events.

Parameters

<i>port</i>	DP port index
-------------	---------------

Returns

true if HPD is connected, false otherwise

6.1.3.7 eval_app_alt_hw_cmd()

```
bool eval_app_alt_hw_cmd (
    uint8_t port,
    uint8_t * cmd_param )
```

This function evaluates received application HW command.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cmd_param</i>	Pointer to received application HW command data.

Returns

true if APP command passed successful, false if APP command is invalid or contain unacceptable fields.

6.1.3.8 eval_hpdu_cmd()

```
bool eval_hpdu_cmd (
    uint8_t port,
    uint32_t cmd )
```

This function evaluates received HPD application command.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cmd</i>	Pointer to received HPD application command data.

Returns

true if HPD APP command passed successful, false if HPD APP command is invalid or contain unacceptable fields.

6.1.3.9 eval_mux_cmd()

```
bool eval_mux_cmd (
    uint8_t port,
    uint32_t cmd )
```

This function evaluates received MUX application command.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cmd</i>	Pointer to received MUX application command data.

Returns

true if MUX APP command passed successful, false if MUX APP command is invalid or contain unacceptable fields.

6.1.3.10 get_mux_state()

```
mux_select_t get_mux_state (
    uint8_t port )
```

Get the current state of the MUX.

Parameters

<i>port</i>	PD port index.
-------------	----------------

Returns

Active MUX setting.

6.1.3.11 set_mux()

```
bool set_mux (
    uint8_t port,
    mux_select_t cfg,
    uint32_t custom_data )
```

This function sets appropriate MUX configuration based on the function parameters.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cfg</i>	Required MUX configuration.
<i>custom_data</i>	Additional data in case of custom AR MUX configuration.

Returns

true if MUX set passed successful, false if MUX setting is invalid or contain unacceptable fields.

6.2 app/alt_mode/alt_modes_mngr.h File Reference

```
#include <stdint.h>
#include <pd.h>
#include <config.h>
#include <vdm_task_mngr.h>
```

Data Structures

- union `alt_mode_evt_t`
- struct `alt_mode_evt_t::ALT_MODE_EVT`
- struct `alt_mode_evt_t::ALT_MODE_EVT_DATA`
- struct `comp_tbl_t`
- struct `alt_mode_reg_info_t`
- struct `alt_mode_info_t`

Macros

- #define `ATCH_TGT` (1u)
- #define `CABLE` (2u)
- #define `MODE_NOT_SUPPORTED` (0xFFu)
- #define `EMPTY_VDO` (0x00000000u)
- #define `NONE_VDO` (0xFFFFFFFFu)
- #define `MAX_SUPP_ALT_MODES` (4u)
- #define `VDO_START_IDX` (1u)
- #define `NONE_MODE_MASK` (0x00000000u)
- #define `FULL_MASK` (0xFFFFFFFFu)
- #define `EN_FLAG_MASK` (0x00000001u)
- #define `EXIT_ALL_MODES` (0x7u)
- #define `CBL_DIR_SUPP_MASK` (0x780u)
- #define `USB_2_0_SUPP` (0u)
- #define `USB_GEN_1_SUPP` (1u)
- #define `USB_GEN_2_SUPP` (2u)
- #define `ALT_MODE_EVT_SIZE` (2u)
- #define `ALT_MODE_EVT_IDX` (0u)
- #define `ALT_MODE_EVT_DATA_IDX` (1u)
- #define `NO_DATA` (0u)
- #define `MAX_RETRY_CNT` (9u)
- #define `SET_FLAG`(status, bit_idx) ((status) |= (1 << ((uint32_t)(bit_idx))))
- #define `REMOVE_FLAG`(status, bit_idx) ((status) &= (~1 << ((uint32_t)(bit_idx))))
- #define `IS_FLAG_CHECKED`(status, bit_idx) (((status) >> (uint32_t)(bit_idx)) & `EN_FLAG_MASK`)
- #define `VDM_HDR` vdm_header.std_vdm_hdr

Typedefs

- typedef void(* `alt_mode_cbk_t`) (uint8_t port)
- typedef bool(* `alt_mode_app_cbk_t`) (uint8_t port, `alt_mode_evt_t` app_cmd)

Enumerations

- enum `alt_mode_mngr_state_t`{ `ALT_MODE_MNGR_STATE_DISC_MODE` = 0, `ALT_MODE_MNGR_STATE_PROCESS` }
- enum `alt_mode_state_t`{
 `ALT_MODE_STATE_DISABLE` = 0, `ALT_MODE_STATE_IDLE`, `ALT_MODE_STATE_INIT`, `ALT_MODE_STATE_SEND`, `ALT_MODE_STATE_WAIT_FOR_RESP`, `ALT_MODE_STATE_FAIL`, `ALT_MODE_STATE_RUN`, `ALT_MODE_STATE_EXIT` }
- enum `alt_mode_app_evt_t`{
 `AM_NO_EVT` = 0, `AM_EVT_SVID_NOT_FOUND`, `AM_EVT_ALT_MODE_ENTERED`, `AM_EVT_ALT_MODE_EXITED`, `AM_EVT_DISC_FINISHED`, `AM_EVT_SVID_NOT_SUPP`, `AM_EVT_SVID_SUPP`, `AM_EVT_ALT_MODE_SUPP`, `AM_EVT_SOP_RESP_FAILED`, `AM_EVT_CBL_RESP_FAILED`, `AM_EVT_CBL_NOT_SUPP_ALT_MODE`, `AM_EVT_NOT_SUPP_PARTNER_CAP`, `AM_EVT_DATA_EVT` }
- enum `alt_mode_app_cmd_t`{
 `AM_NO_CMD` = 0, `AM_SET_TRIGGER_MASK`, `AM_CMD_ENTER` = 3, `AM_CMD_EXIT`, `AM_CMD_SPEC` }
- enum `fail_status_t` { `BUSY` = 0, `GOOD_CRC_NOT_RSVD`, `TIMEOUT`, `NACK` }

Functions

- `vdm_task_t reg_alt_mode_mngr` (`uint8_t port`, `atch_tgt_info_t *atch_tgt_info`, `vdm_msg_info_t *vdm_msg_info`)
- `vdm_task_t vdm_task_mngr_alt_mode_process` (`uint8_t port`, `vdm_evt_t vdm_evt`)
- `bool eval_rec_vdm` (`uint8_t port`, `const pd_packet_t *vdm`)
- `const uint32_t * form_alt_mode_event` (`uint8_t port`, `uint16_t svid`, `uint8_t am_idx`, `alt_mode_app_evt_t evt`, `uint32_t data`)
- `bool eval_app_alt_mode_cmd` (`uint8_t port`, `uint8_t *cmd`, `uint8_t *data`)
- `bool is_alt_mode_mngr_idle` (`uint8_t port`)
- `void alt_mode_mngr_sleep` (`uint8_t port`)
- `void alt_mode_mngr_wakeup` (`uint8_t port`)
- `void reset_alt_mode_info` (`alt_mode_info_t *info`)
- `dpm_pd_cmd_buf_t * get_vdm_buff` (`uint8_t port`)
- `uint8_t is_svid_supported` (`uint16_t svid`, `uint8_t port`)
- `uint8_t alt_mode_get_status` (`uint8_t port`)
- `void alt_mode_mngr_exit_all` (`uint8_t port`)
- `uint8_t get_alt_mode_numb` (`uint8_t port`)

6.2.1 Detailed Description

Alternate Mode Manager header file.

6.2.2 Typedef Documentation

6.2.2.1 alt_mode_app_cbk_t

```
typedef bool(* alt_mode_app_cbk_t) (uint8_t port, alt_mode_evt_t app_cmd)
```

This type of the function is used by alt modes manager to run alternative mode analisys of received APP command

Parameters

<i>port</i>	Port index the function is performed for.
<i>hpi_cmd</i>	Received APP command data.

Returns

true if APP command passed successful, false if APP command is invalid or contain unacceptable fields.

6.2.2.2 alt_mode_cbk_t

```
typedef void(* alt_mode_cbk_t) (uint8_t port)
```

This type of the function is used by alt modes manager to communicate with any of supported alt modes.

Parameters

<i>port</i>	Port index the function is performed for .
-------------	--

Returns

None.

6.2.3 Enumeration Type Documentation**6.2.3.1 alt_mode_app_cmd_t**

```
enum alt_mode_app_cmd_t
```

This enumeration holds all possible APP command related to Alt modes handling.

Enumerator

AM_NO_CMD	Empty command.
AM_SET_TRIGGER_MASK	Sets trigger mask to prevent auto-entering of the selected alternative modes.
AM_CMD_ENTER	Enter to selected alternative mode.
AM_CMD_EXIT	Exit from selected alternative mode.
AM_CMD_SPEC	Specific alternative EC mode command with data.

6.2.3.2 alt_mode_app_evt_t

```
enum alt_mode_app_evt_t
```

This enumeration holds all possible application events related to Alt modes handling.

Enumerator

AM_NO_EVT	Empty event.
-----------	--------------

Enumerator

AM_EVT_SVID_NOT_FOUND	Sends to EC if UFP doesn't support any SVID.
AM_EVT_ALT_MODE_ENTERED	Alternative mode entered.
AM_EVT_ALT_MODE_EXITED	Alternative mode exited.
AM_EVT_DISC_FINISHED	Discovery process was finished.
AM_EVT_SVID_NOT_SUPP	CCGx doesn't support received SVID.
AM_EVT_SVID_SUPP	CCGx supports received SVID.
AM_EVT_ALT_MODE_SUPP	CCGx supports alternate mode.
AM_EVT_SOP_RESP_FAILED	UFP VDM response failed.
AM_EVT_CBL_RESP_FAILED	Cable response failed.
AM_EVT_CBL_NOT_SUPP_ALT_MODE	Cable capabilities couldn't provide alternative mode hanling.
AM_EVT_NOT_SUPP_PARTNER_CAP	CCGx and UFP capabilities not consistent.
AM_EVT_DATA_EVT	Specific alternative mode event with data.

6.2.3.3 alt_mode_mngr_state_t

```
enum alt_mode_mngr_state_t
```

This enumeration holds all possible Alt modes manager states when DFP.

Enumerator

ALT_MODE_MNGR_STATE_DISC_MODE	Alt modes manager discovery mode state.
ALT_MODE_MNGR_STATE_PROCESS	Alt modes manager alternative modes processing state.

6.2.3.4 alt_mode_state_t

```
enum alt_mode_state_t
```

This enumeration holds all possible states of each alt mode which is handled by Alt modes manager.

Enumerator

ALT_MODE_STATE_DISABLE	State when alternative mode functionality is disabled.
ALT_MODE_STATE_IDLE	State when alternative mode is idle.
ALT_MODE_STATE_INIT	State when alternative mode initiate its functionality.
ALT_MODE_STATE_SEND	State when alternative mode needs to send VDM message.
ALT_MODE_STATE_WAIT_FOR_RESP	State while alternative mode wait for VDM response.
ALT_MODE_STATE_FAIL	State when alternative mode VDM response fails.
ALT_MODE_STATE_RUN	State when alternative mode need to be running.
ALT_MODE_STATE_EXIT	State when alternative mode exits and resets all related variables.

6.2.3.5 fail_status_t

```
enum fail_status_t
```

Enum of the VDM failure response status.

Enumerator

BUSY	Target is BUSY.
GOOD_CRC_NOT_RSVD	Good CRC wasn't received.
TIMEOUT	No response or corrupted packet.
NACK	Sent VDM NACKed.

6.2.4 Function Documentation

6.2.4.1 alt_mode_get_status()

```
uint8_t alt_mode_get_status (
    uint8_t port )
```

Retrieve the current alternate mode status for a PD port.

Parameters

<i>port</i>	PD port to be queried.
-------------	------------------------

Returns

The alternate mode status.

6.2.4.2 alt_mode_mngr_exit_all()

```
void alt_mode_mngr_exit_all (
    uint8_t port )
```

Exit all active alternate modes.

Parameters

<i>port</i>	PD port index.
-------------	----------------

Returns

None

6.2.4.3 alt_mode_mngr_sleep()

```
void alt_mode_mngr_sleep (
```

```
    uint8_t port )
```

Prepare the alt modes manager for device deep sleep.

Parameters

<i>port</i>	Index of USB-PD port to be prepared for sleep.
-------------	--

Returns

None

6.2.4.4 alt_mode_mngr_wakeup()

```
void alt_mode_mngr_wakeup (
    uint8_t port )
```

Restore the alt modes manager state after waking from deep sleep.

Parameters

<i>port</i>	Index of USB-PD port to be restored.
-------------	--------------------------------------

Returns

None

6.2.4.5 eval_app_alt_mode_cmd()

```
bool eval_app_alt_mode_cmd (
    uint8_t port,
    uint8_t * cmd,
    uint8_t * data )
```

This function analises, parses and run alternative mode analisys function if received command is specific alt mode command.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cmd</i>	Pointer to received alt mode APP command.
<i>data</i>	Pointer to received alt mode APP command additional data.

Returns

true if APP command passed successful, false if APP command is invalid or contain unacceptable fields.

6.2.4.6 eval_rec_vdm()

```
bool eval_rec_vdm (
    uint8_t port,
    const pd_packet_t * vdm )
```

This function run received VDM analisys if CCG is UFP.

Parameters

<i>port</i>	Port index the function is performed for.
<i>vdm</i>	Pointer to pd packet which contains received VDM.

Returns

true if received VDM could handled successful and VDM response need to be sent with ACK. If received VDM should be NACKed then returns false

6.2.4.7 form_alt_mode_event()

```
const uint32_t* form_alt_mode_event (
    uint8_t port,
    uint16_t svid,
    uint8_t am_idx,
    alt_mode_app_evt_t evt,
    uint32_t data )
```

Fill alt mode APP event fields with appropriate values.

Parameters

<i>port</i>	Port index the event is performed for.
<i>svid</i>	SVID of alternative mode which event refers to.
<i>am_idx</i>	Index of alternative mode in compatibility table which event refers to.
<i>evt</i>	Alternative mode APP event.
<i>data</i>	Alternative mode APP event data.

Returns

pointer to the event related data.

6.2.4.8 get_alt_mode_numb()

```
uint8_t get_alt_mode_numb (
    uint8_t port )
```

Get number of the alternate modes for choosen port.

Parameters

<i>port</i>	PD port index.
-------------	----------------

Returns

Number of the alternates modes for the chosen port and current data role.

6.2.4.9 get_vdm_buff()

```
dpm_pd_cmd_buf_t* get_vdm_buff (
    uint8_t port )
```

Returns pointer to VDM buffer.

Parameters

<i>port</i>	Index of Type-C port.
-------------	-----------------------

Returns

Pointer to VDM buffer.

6.2.4.10 is_alt_mode_mngr_idle()

```
bool is_alt_mode_mngr_idle (
    uint8_t port )
```

Check whether the DFP VDM manager for the selected port is idle.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

true if manager is busy, false - if idle.

6.2.4.11 is_svid_supported()

```
uint8_t is_svid_supported (
    uint16_t svid,
    uint8_t port )
```

Check for presence of alt modes for given svid in alt modes compatibility table.

Parameters

<i>svid</i>	Received SVID.
<i>port</i>	Index of Type-C port.

Returns

True if SVID is supported by CCG.

6.2.4.12 reg_alt_mode_mngr()

```
vdm_task_t reg_alt_mode_mngr (
    uint8_t port,
    attach_tgt_info_t * attach_tgt_info,
    vdm_msg_info_t * vdm_msg_info )
```

This function register pointers to attached dev/ama/cable info and run alt modes manager if success.

Parameters

<i>port</i>	Port index the function is performed for.
<i>attach_tgt_info</i>	Pointer to struct which holds discovery info about attached targets.
<i>vdm_msg_info</i>	Pointer to struct which holds info of received/sent VDM

Returns

VDM manager task VDM_TASK_ALT_MODE if CCG support any alternative mode. If CCG doesn't support alternative modes function returns VDM_TASK_EXIT.

6.2.4.13 reset_alt_mode_info()

```
void reset_alt_mode_info (
    alt_mode_info_t * info )
```

Resets alternative mode command info structure.

Parameters

<i>info</i>	Pointer to alternative mode info structure.
-------------	---

Returns

None.

6.2.4.14 vdm_task_mngr_alt_mode_process()

```
vdm_task_t vdm_task_mngr_alt_mode_process (
    uint8_t port,
    vdm_evt_t vdm_evt )
```

This function uses by DFP VDM manager to run alt modes manager processing.

Parameters

<i>port</i>	Port index the function is performed for.
<i>vdm_evt</i>	Current DFP VDM manager event.

Returns

DFP VDM manager task based on alt modes manager processing results.

6.3 app/alt_mode/dp_sid.h File Reference

```
#include <pd.h>
#include <alt_modes_mngr.h>
```

Macros

- #define DP_SVID (0xFF01u)
- #define DP_ALT_MODE_ID (0u)
- #define MAX_DP_VDO_NUMB (1u)
- #define DP_VDO_IDX (0u)
- #define STATUS_UPDATE_VDO (0x01u)
- #define DP_1_3_SIGNALING (0x0001u)
- #define DP_CONFIG_SELECT (2u)
- #define USB_CONFIG_SELECT (0u)
- #define DP_USB_SS_CONFIG (0b00000000u)
- #define DP_DFP_D_CONFIG_C (0b00000100u)
- #define DP_DFP_D_CONFIG_D (0b00001000u)
- #define DP_DFP_D_CONFIG_E (0b00010000u)
- #define DP_DFP_D_CONFIG_F (0b00100000u)
- #define DP_INVALID_CFG (0b11111111u)
- #define HPD_LOW_IRQ_LOW (0x0u)
- #define HPD_HIGH_IRQ_LOW (0x1u)
- #define HPD_LOW_IRQ_HIGH (0x2u)
- #define HPD_HIGH_IRQ_HIGH (0x3u)
- #define DP_QUEUE_STATE_SIZE (2u)
- #define DP_HPD_STATE_MASK (0x0003u)
- #define DP_QUEUE_EMPTY_INDEX (0x0u)
- #define DP_QUEUE_FULL_INDEX (7u)
- #define DP_UFP_MAX_QUEUE_SIZE (4u)
- #define GET_HPD_IRQ_STAT(status) (((status) >> 7u)& 0x3)
- #define DP_SINK_CTRL_CMD (3u)
- #define DP_MUX_CTRL_CMD (1u)
- #define DP_APP_CFG_CMD (2u)
- #define DP_APP_CFG_USB_IDX (6u)
- #define DP_APP_CFG_CMD_MAX_NUMB (6u)
- #define DP_ALLOWED_MUX_CONFIG_EVT (1u)
- #define DP_STATUS_UPDATE_EVT (2u)
- #define DP_CFG_CMD_ACK_MASK (0x200)

Enumerations

- enum dp_state_t {
 DP_STATE_IDLE = 0, DP_STATE_ENTER = 4, DP_STATE_STATUS_UPDATE = 16, DP_STATE_CONFIG = 17,
 DP_STATE_ATT = 6, DP_STATE_EXIT = 5
 }
- enum dp_port_cap_t { DP_PORT_CAP_RSVD = 0, DP_PORT_CAP_UFP_D, DP_PORT_CAP_DFP_D,
 DP_PORT_CAP_BOTH }

- enum `dp_conn_t` { `DP_CONN_NONE` = 0, `DP_CONN_DFP_D`, `DP_CONN_UFP_D`, `DP_CONN_BOTH` }
- enum `dp_stat_bm_t` {
`DP_STAT_DFP_CONN` = 0, `DP_STAT_UFP_CONN`, `DP_STAT_PWR_LOW`, `DP_STAT_EN`,
`DP_STAT_MF`, `DP_STAT_USB_CNFG`, `DP_STAT_EXIT`, `DP_STAT_HPD`,
`DP_STAT_IRQ` }

Functions

- `alt_mode_info_t * reg_dp_modes` (uint8_t port, `alt_mode_reg_info_t *reg_info`)

6.3.1 Detailed Description

DisplayPort alternate mode handler header file.

6.3.2 Enumeration Type Documentation

6.3.2.1 `dp_conn_t`

enum `dp_conn_t`

This enumeration holds possible DFP_D/UFP_D Connected status (Status Update Message).

Enumerator

<code>DP_CONN_NONE</code>	Neither DFP_D nor UFP_D is connected.
<code>DP_CONN_DFP_D</code>	DFP_D is connected.
<code>DP_CONN_UFP_D</code>	UFP_D is connected.
<code>DP_CONN_BOTH</code>	Both DFP_D and UFP_D are connected.

6.3.2.2 `dp_port_cap_t`

enum `dp_port_cap_t`

This enumeration holds possible DP capabilities.

Enumerator

<code>DP_PORT_CAP_RSVD</code>	Reserved capability.
<code>DP_PORT_CAP_UFP_D</code>	UFP is UFP_D-capable.
<code>DP_PORT_CAP_DFP_D</code>	UFP is DFP_D-capable.
<code>DP_PORT_CAP_BOTH</code>	UFP is DFP_D and UFP-D capable.

6.3.2.3 dp_stat_bm_t

enum `dp_stat_bm_t`

This enumeration holds corresponding bit positions of Status Update VDM fields.

Enumerator

<code>DP_STAT_DFP_CONN</code>	DFP_D is connected field bit position.
<code>DP_STAT_UFP_CONN</code>	UFP_D is connected field bit position.
<code>DP_STAT_PWR_LOW</code>	Power Low field bit position.
<code>DP_STAT_EN</code>	Enabled field bit position.
<code>DP_STAT_MF</code>	Multi-function Preferred field bit position.
<code>DP_STAT_USB_CNFG</code>	USB Configuration Request field bit position.
<code>DP_STAT_EXIT</code>	Exit DP Request field bit position.
<code>DP_STAT_HPD</code>	HPD state field bit position.
<code>DP_STAT_IRQ</code>	HPD IRQ field bit position.

6.3.2.4 dp_state_t

enum `dp_state_t`

This enumeration holds all possible DP states.

Enumerator

<code>DP_STATE_IDLE</code>	Idle state.
<code>DP_STATE_ENTER</code>	Enter mode state.
<code>DP_STATE_STATUS_UPDATE</code>	DP Status Update state.
<code>DP_STATE_CONFIG</code>	DP Configure state.
<code>DP_STATE_ATT</code>	DP Attention state.
<code>DP_STATE_EXIT</code>	Exit mode state.

6.3.3 Function Documentation

6.3.3.1 reg_dp_modes()

```
alt_mode_info_t* reg_dp_modes (
    uint8_t port,
    alt_mode_reg_info_t * reg_info )
```

This function analyses Discovery information to find out if further DP alternative mode processing is allowed.

Parameters

<code>port</code>	Port index the function is performed for.
<code>reg_info</code>	Pointer to structure which holds alt mode register info.

Returns

Pointer to DP alternative mode command structure if analysis passed successful. In other case function returns NULL pointer

6.4 app/alt_mode/intel_ridge.h File Reference

```
#include <alt_mode_hw.h>
#include <status.h>
#include <hpd.h>
```

Data Structures

- union `ridge_reg_t`
- struct `ridge_reg_t::USBPD_STATUS_REG`
- struct `ridge_reg_t::USBPD_CMD_REG`

Macros

- #define `RIDGE_DISCON_STATE_MASK` (0x00)
- #define `RIDGE_DATA_CONN_MASK` (0x10171)
- #define `RIDGE_USB_STATE_MASK` (0x31)
- #define `RIDGE_SAFE_STATE_MASK` (0x01)
- #define `RIDGE_DP_4_LANE_MASK` (0x111)
- #define `RIDGE_DP_2_LANE_MASK` (0x531)
- #define `DP_PIN_CONFIG_C` (0b00000100u)
- #define `RIDGE_TBT_MODE_MASK` (0x10001)
- #define `RIDGE_DEBUG_MODE_MASK` (0x11)
- #define `TBT_HOST_CONN_MASK` (0x01)
- #define `TR_IRQ_ACK_MASK` (0x2000)
- #define `TR_HPD_IRQ_MASK` (0x4000)
- #define `TR_HPD_LVL_MASK` (0x8000)
- #define `TR_USB_HOST_CONN_MASK` (0x10)
- #define `TR_DP_HOST_CONN_MASK` (0x20)
- #define `RIDGE_STATUS_OCP_MASK` (0x08)

Typedefs

- typedef void(* `ridge_ctrl_change_cb_t`) (uint8_t port)

Functions

- bool `ridge_set_mux` (uint8_t port, `mux_select_t` mux_cfg, uint8_t polarity, uint32_t cfg)
- `ccg_status_t tr_hpd_init` (uint8_t port, `hpd_event_cbk_t` cbk)
- void `tr_hpd_deinit` (uint8_t port)
- `ccg_status_t tr_hpd_sendevt` (uint8_t port, `hpd_event_type_t` evtype)
- void `ridge_eval_cmd` (uint8_t port, uint32_t stat, uint32_t stat_mask)
- bool `tr_is_hpd_change` (uint8_t port)
- void `ridge_set_ctrl_change_cb` (uint8_t port, `ridge_ctrl_change_cb_t` cb)

6.4.1 Detailed Description

Intel Alpine/Titan Ridge control interface header file.

6.4.2 Function Documentation

6.4.2.1 ridge_eval_cmd()

```
void ridge_eval_cmd (
    uint8_t port,
    uint32_t stat,
    uint32_t stat_mask )
```

Analyses received ridge command register content.

Parameters

<i>port</i>	USB-PD port index corresponding to the status update.
<i>stat</i>	Command register value.
<i>stat_mask</i>	Bit mask of TR command register which were changed from the previous time

Returns

true if the interface is idle, false otherwise.

6.4.2.2 ridge_set_ctrl_change_cb()

```
void ridge_set_ctrl_change_cb (
    uint8_t port,
    ridge_ctrl_change_cb_t cb )
```

Register a callback for notification of data control register changes.

Parameters

<i>port</i>	PD port index.
<i>cb</i>	Pointer to callback function.

Returns

None

6.4.2.3 ridge_set_mux()

```
bool ridge_set_mux (
    uint8_t port,
    mux_select_t mux_cfg,
```

```
    uint8_t polarity,
    uint32_t cfg )
```

This function set AR/TR registers in accordance to input parameters.

Parameters

<i>port</i>	Port index the AR/TR settings are performed for.
<i>mux_cfg</i>	MUX configuration.
<i>polarity</i>	Attached target Type-C Polarity.
<i>cfg</i>	Contains AR/TR register settings in case of TBT alt mode is active.

Returns

true if AR/TR was set successful, in the other case - false

6.4.2.4 tr_hpd_deinit()

```
void tr_hpd_deinit (
    uint8_t port )
```

Disables Titan Ridge the HPD functionality for the specified PD port.

Parameters

<i>port</i>	PD port index. Caller should ensure to provide only valid values.
-------------	---

Returns

None.

6.4.2.5 tr_hpd_init()

```
ccg_status_t tr_hpd_init (
    uint8_t port,
    hpd_event_cbk_t cbk )
```

Enables Titan Ridge the HPD functionality for the specified PD port.

Parameters

<i>port</i>	PD port index. Caller should ensure to provide only valid values.
<i>cbk</i>	callback to be used for command completion event.

Returns

Returns CCG_STAT_SUCCESS in case of success, error code otherwise.

6.4.2.6 tr_hpd_sendevt()

```
ccg_status_t tr_hpd_sendevt (
    uint8_t port,
    hpd_event_type_t evtype )
```

Send the desired HPD event out through the Titan Ridge HPD GPIO. Only the HPD_EVENT_UNPLUG, HPD_EVENT_UNPLUG and HPD_EVENT_IRQ events should be requested.

Parameters

<i>port</i>	Port on which HPD event is to be sent.
<i>evtype</i>	Type of HPD event to be sent.

Returns

Returns CCG_STAT_SUCCESS in case of success, error code otherwise.

6.4.2.7 tr_is_hpd_change()

```
bool tr_is_hpd_change (
    uint8_t port )
```

Indicates is HPD level changed from the previous state.

Parameters

<i>port</i>	USB-PD port index corresponding to the status update.
-------------	---

Returns

None.

6.5 app/alt_mode/intel_vid.h File Reference

```
#include <pd.h>
#include <alt_modes_mngr.h>
```

Macros

- #define **INTEL_VID** (0x8087u)
- #define **TBT_ALT_MODE_ID** (1u)
- #define **MAX_TBT_VDO_NUMB** (1u)
- #define **TBT_VDO_IDX** (0u)
- #define **GET_LEGACY_TBT_ADAPTER**(status) ((status >> 16) & 0x1)
- #define **TBT_EXIT**(status) ((status >> 4) & 0x1)
- #define **BB_STATUS**(status) ((status >> 3) & 0x1)
- #define **USB2_ENABLE**(status) ((status >> 2) & 0x1)

Enumerations

- enum `tbt_state_t` { `TBT_STATE_IDLE` = 0, `TBT_STATE_ENTER` = 4, `TBT_STATE_ATT` = 6, `TBT_STATE_EXIT` = 5 }

Functions

- `alt_mode_info_t * reg_intel_modes` (`uint8_t port`, `alt_mode_reg_info_t *reg_info`)

6.5.1 Detailed Description

Thunderbolt (Intel VID) alternate mode handler header file.

6.5.2 Enumeration Type Documentation

6.5.2.1 `tbt_state_t`

```
enum tbt_state_t
```

This enumeration holds all possible TBT states.

Enumerator

<code>TBT_STATE_IDLE</code>	Idle state.
<code>TBT_STATE_ENTER</code>	Enter mode state.
<code>TBT_STATE_ATT</code>	Attention state.
<code>TBT_STATE_EXIT</code>	Exit mode state.

6.5.3 Function Documentation

6.5.3.1 `reg_intel_modes()`

```
alt_mode_info_t* reg_intel_modes (
    uint8_t port,
    alt_mode_reg_info_t * reg_info )
```

This function analyses Discovery information to find out if further TBT alternative mode processing is allowed.

Parameters

<code>port</code>	Port index the function is performed for.
<code>reg_info</code>	Pointer to structure which holds alt mode register info.

Returns

Pointer to TBT alternative mode command structure if analysis passed successful. In other case function returns NULL pointer

6.6 app/alt_mode/vdm_task_mngr.h File Reference

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

Data Structures

- struct `atch_tgt_info_t`
- struct `vdm_msg_info_t`

Macros

- #define `PD_SVID_ID_HDR_VDO_START_IDX` (4u)
- #define `PD_DISC_ID_AMA_VDO_IDX` (3u)
- #define `MAX_SVID_VDO_SUPP` (32u)
- #define `MAX_CABLE_SVID_SUPP` (4u)
- #define `STD_SVID` (0xFF00u)
- #define `MAX_DISC_SVID_COUNT` (10u)

Enumerations

- enum `vdm_task_t` {
 `VDM_TASK_WAIT` = 0, `VDM_TASK_INIT`, `VDM_TASK_DISC_ID`, `VDM_TASK_DISC_SVID`,
 `VDM_TASK_REG_ATCH_TGT_INFO`, `VDM_TASK_EXIT`, `VDM_TASK_SEND_MSG`, `VDM_TASK_ALT_MODE`
}
- enum `vdm_evt_t` { `VDM_EVT_RUN` = 0, `VDM_EVT_EVAL`, `VDM_EVT_FAIL`, `VDM_EVT_EXIT` }

Functions

- void `enable_vdm_task_mngr` (uint8_t port)
- void `vdm_task_mngr` (uint8_t port)
- void `vdm_task_mngr_deinit` (uint8_t port)
- bool `is_vdm_task_idle` (uint8_t port)
- uint8_t * `vdm_get_disc_id_resp` (uint8_t port, uint8_t *resp_len_p)
- uint8_t * `vdm_get_disc_svid_resp` (uint8_t port, uint8_t *resp_len_p)

6.6.1 Detailed Description

VDM task manager header file.

6.6.2 Enumeration Type Documentation

6.6.2.1 vdm_evt_t

enum `vdm_evt_t`

VDM manager events definition.

This enumeration lists the various VDM mngr events to handle VDMs.

Enumerator

VDM_EVT_RUN	This event is responsible for running any of DFP VDM manager task .
VDM_EVT_EVAL	This event is responsible for evaluating VDM response .
VDM_EVT_FAIL	This event notifies task manager task if VDM response fails .
VDM_EVT_EXIT	This event runs exiting from VDM task manager task .

6.6.2.2 vdm_task_t

```
enum vdm_task_t
```

VDM manager tasks definition.

This enumeration lists the various VDM mngr tasks to handle VDMs.

Enumerator

VDM_TASK_WAIT	DFP manager wait task while waiting for VDM response.
VDM_TASK_INIT	This task is responsible for initializing of VDM manager.
VDM_TASK_DISC_ID	This task is responsible for VDM Discovery ID flow.
VDM_TASK_DISC_SVID	This task is responsible for VDM Discovery SVID flow.
VDM_TASK_REG_ATCH_TGT_INFO	This task is responsible for registering of Discovery result information in alt mode manager.
VDM_TASK_EXIT	This task deinitializes VDM task manager.
VDM_TASK_SEND_MSG	This task is responsible for forming and sending VDM message .
VDM_TASK_ALT_MODE	This task is responsible for running of alt mode manager .

6.6.3 Function Documentation

6.6.3.1 enable_vdm_task_mngr()

```
void enable_vdm_task_mngr (
    uint8_t port )
```

Enables VDM task mngr functionality.

Parameters

port	PD port corresponding to the VDM task manager.
------	--

Returns

None.

6.6.3.2 is_vdm_task_idle()

```
bool is_vdm_task_idle (
    uint8_t port )
```

Check whether the VDM task for the port is idle.

Parameters

<i>port</i>	PD port corresponding to the VDM task manager.
-------------	--

Returns

None.

6.6.3.3 vdm_get_disc_id_resp()

```
uint8_t* vdm_get_disc_id_resp (
    uint8_t port,
    uint8_t * resp_len_p )
```

Obtain the last DISC_ID response received by the CCG device from a port partner.

Parameters

<i>port</i>	PD port index.
<i>resp_len_p</i>	Length of response in PD data objects.

Returns

Pointer to the actual response data.

6.6.3.4 vdm_get_disc_svid_resp()

```
uint8_t* vdm_get_disc_svid_resp (
    uint8_t port,
    uint8_t * resp_len_p )
```

Obtain the collective DISC_SVID response received by the CCG device from a port partner.

Parameters

<i>port</i>	PD port index.
<i>resp_len_p</i>	Length of response in PD data objects.

Returns

Pointer to the actual response data.

6.6.3.5 vdm_task_mngr()

```
void vdm_task_mngr (
    uint8_t port )
```

Main VDM task mngr function.

Parameters

<i>port</i>	PD port corresponding to the VDM task manager.
-------------	--

Returns

None.

6.6.3.6 vdm_task_mngr_deinit()

```
void vdm_task_mngr_deinit (
    uint8_t port )
```

This function deinitializes VDM task manager and resets all related variables.

Parameters

<i>port</i>	PD port corresponding to the VDM task manager.
-------------	--

Returns

None.

6.7 app/app.h File Reference

```
#include <stdint.h>
#include <pd.h>
#include <pdss_hal.h>
#include <alt_mode_hw.h>
```

Data Structures

- struct [app_status_t](#)

Macros

- #define APP_PSOURCE_EN_TIMER (APP_TIMERS_START_ID)
- #define APP_PSOURCE_EN_MONITOR_TIMER (31u)
- #define APP_PSOURCE_EN_HYS_TIMER (32u)
- #define APP_PSOURCE_DIS_TIMER (33u)
- #define APP_PSOURCE_DIS_MONITOR_TIMER (34u)
- #define APP_PSOURCE_CF_TIMER (35u)
- #define APP_PSOURCE_CF_TIMER_PERIOD (100u)

- #define APP_PSOURCE_DIS_EXT_DIS_TIMER (36u)
- #define APP_PSOURCE_DIS_EXT_DIS_TIMER_PERIOD (10u)
- #define APP_DB_SNK_FET_DIS_DELAY_TIMER (37u)
- #define APP_PSINK_DIS_TIMER (38u)
- #define APP_PSINK_DIS_TIMER_PERIOD (250u)
- #define APP_PSINK_DIS_VBUS_IN_DIS_PERIOD (20u)
- #define APP_PSINK_DIS_MONITOR_TIMER (39u)
- #define APP_PSINK_DIS_MONITOR_TIMER_PERIOD (1u)
- #define APP_VDM_BUSY_TIMER (40u)
- #define APP_AME_TIMEOUT_TIMER (41u)
- #define APP_VBUS_OCP_OFF_TIMER (42u)
- #define APP_VBUS_OVP_OFF_TIMER (43u)
- #define APP_VBUS_UVP_OFF_TIMER (44u)
- #define APP_VBUS_SCP_OFF_TIMER (45u)
- #define APP_FAULT_RECOVERY_TIMER (46u)
- #define APP_FAULT_RECOVERY_TIMER_PERIOD (100u)
- #define APP_FAULT_RECOVERY_MAX_WAIT (500u)
- #define APP_SBU_DELAYED_CONNECT_TIMER (47u)
- #define APP_SBU_DELAYED_CONNECT_PERIOD (25u)
- #define APP_SBU_TBT_CONNECT_DELAY (50u)
- #define APP_MUX_DELAY_TIMER (48u)
- #define APP_MUX_POLL_TIMER (49u)
- #define APP_VDM_BUSY_TIMER_PERIOD (50u)
- #define APP_VDM_FAIL_RETRY_PERIOD (100u)
- #define APP_CABLE_POWER_UP_DELAY (55u)
- #define APP_CABLE_VDM_START_DELAY (5u)
- #define APP_AME_TIMEOUT_TIMER_PERIOD (800u)
- #define APP_DB_SNK_FET_DIS_DELAY_TIMER_PERIOD (50u)
- #define APP_BB_ON_TIMER (60u)
- #define APP_BB_ON_TIMER_PERIOD (250u)
- #define APP_BB_OFF_TIMER (61u)
- #define APP_INITIATE_SEND_IRQ_CLEAR_ACK (64u)
- #define APP_INITIATE_SEND_IRQ_CLEAR_ACK_PERIOD (1u)
- #define APP_BC_GENERIC_TIMER1 (70u)
- #define APP_BC_GENERIC_TIMER2 (71u)
- #define APP_BC_DP_DM_DEBOUNCE_TIMER (72u)
- #define APP_VDM_NOT_SUPPORT_RESP_TIMER_ID (73u)
- #define APP_VDM_NOT_SUPPORT_RESP_TIMER_PERIOD (5)
- #define CCG_ACTIVITY_TIMER_ID (81u)
- #define CCG_ACTIVITY_TIMER_PERIOD (500)
- #define OTP_DEBOUNCE_TIMER_ID (82u)
- #define OTP_DEBOUNCE_PERIOD (1)
- #define TYPE_A_CUR_SENSE_TIMER_ID (83u)
- #define TYPE_A_CUR_SENSE_TIMER_PERIOD (30000)
- #define TYPE_A_REG_SWITCH_TIMER_ID (84u)
- #define TYPE_A_REG_SWITCH_TIMER_PERIOD (10)
- #define TYPE_A_PWM_STEP_TIMER_ID (85u)
- #define TYPE_A_PWM_STEP_TIMER_PERIOD (1)
- #define PB_DEBOUNCE_TIMER_ID (86u)
- #define PB_DEBOUNCE_PERIOD (1)
- #define APP_PB_VBATT_DEBOUNCE_IN_MS (10)
- #define APP_PSOURCE_VBUS_SET_TIMER (87u)
- #define APP_PSOURCE_VBUS_SET_TIMER_PERIOD (1)
- #define APP_BC_VBUS_CYCLE_TIMER_PERIOD (200u)
- #define APP_BC_SINK_CONTACT_STABLE_TIMER_PERIOD (50u)

- #define APP_BC_DCP_DETECT_TIMER_PERIOD (1100u)
- #define APP_BC_APPLE_DETECT_TIMER_PERIOD (5u)
- #define APP_BC_DP_DM_DEBOUNCE_TIMER_PERIOD (40u)
- #define APP_BC_AFC_DETECT_TIMER_PERIOD (100u)
- #define APP_BC_GLITCH_BC_DONE_TIMER_PERIOD (1500)
- #define APP_BC_GLITCH_DM_HIGH_TIMER_PERIOD (40)
- #define APP_BC_V_NEW_REQUEST_TIMER_PERIOD (200)
- #define ADC_VBUS_MIN_OVP_LEVEL (6500u)

Typedefs

- typedef mux_poll_status_t(* mux_poll_fnc_cbk_t) (uint8_t port)

Enumerations

- enum {
 APP_PORT_FAULT_NONE = 0x00, APP_PORT_VCONN_FAULT_ACTIVE = 0x01, APP_PORT_SINK_FAULT_ACTIVE = 0x02, APP_PORT_SRC_FAULT_ACTIVE = 0x04,
 APP_PORT_VBUS_DROP_WAIT_ACTIVE = 0x08, APP_PORT_DISABLE_IN_PROGRESS = 0x80 }
- enum sys_hw_error_type_t { SYS_HW_ERROR_NONE = 0x00, SYS_HW_ERROR_MUX_ACCESS = 0x01,
 SYS_HW_ERROR_REG_ACCESS = 0x02, SYS_HW_ERROR_BAD_VOLTAGE = 0x04 }
- enum { APP_THERMISTOR_TYPE_NTC = 0x00, APP_THERMISTOR_TYPE_PTC = 0x01, APP_THERMISTOR_TYPE_ERROR = 0x02 }

Functions

- void app_init (void)
- void sln_pd_event_handler (uint8_t port, app_evt_t evt, const void *data)
- uint8_t app_task (uint8_t port)
- app_cbk_t * app_get_callback_ptr (uint8_t port)
- void app_event_handler (uint8_t port, app_evt_t evt, const void *dat)
- app_resp_t * app_get_resp_buf (uint8_t port)
- app_status_t * app_get_status (uint8_t port)
- bool app_sleep (void)
- void app_wakeup (void)
- bool system_sleep (void)
- void vconn_enable (uint8_t port, uint8_t channel)
- void vconn_disable (uint8_t port, uint8_t channel)
- bool vconn_is_present (uint8_t port)
- bool vbus_is_present (uint8_t port, uint16_t volt, int8_t per)
- uint16_t vbus_get_value (uint8_t port)
- void vbus_discharge_on (uint8_t port)
- void vbus_discharge_off (uint8_t port)
- bool system_vconn_ocp_en (uint8_t port, PD_ADC_CB_T cbk)
- void system_vconn_ocp_dis (uint8_t port)
- void app_ovp_enable (uint8_t port, uint16_t volt_mV, bool pfet, PD_ADC_CB_T ovp_cb)
- void app_ovp_disable (uint8_t port, bool pfet)
- void app_upv_enable (uint8_t port, uint16_t volt_mV, bool pfet, PD_ADC_CB_T upv_cb)
- void app_upv_disable (uint8_t port, bool pfet)
- void app_conf_for_faulty_dev_removal (uint8_t port)
- void app_update_bc_src_support (uint8_t port, uint8_t enable)
- void app_update_sys_pwr_state (uint8_t state)
- ccg_status_t app_disable_pd_port (uint8_t port, dpm_typec_cmd_cbk_t cbk)

- bool `app_validate_configurable_offsets` (void)
- bool `mux_ctrl_init` (uint8_t port)
- bool `mux_ctrl_set_cfg` (uint8_t port, `mux_select_t` cfg, uint8_t polarity)
- void `ccg_app_task_init` (void)

6.7.1 Detailed Description

PD application handler header file.

6.7.2 Typedef Documentation

6.7.2.1 `mux_poll_fnc_cbk_t`

```
typedef mux_poll_status_t (* mux_poll_fnc_cbk_t) (uint8_t port)
```

This type of the function is used by app layer to call MUX polling function.

Parameters

<code>port</code>	Port index the function is performed for.
-------------------	---

Returns

MUX polling status

6.7.3 Enumeration Type Documentation

6.7.3.1 anonymous enum

```
anonymous enum
```

Enumerator

<code>APP_PORT_VCONN_FAULT_ACTIVE</code>	Status bit that indicates VConn fault is active.
<code>APP_PORT_SINK_FAULT_ACTIVE</code>	Status bit that indicates sink fault handling is pending.
<code>APP_PORT_SRC_FAULT_ACTIVE</code>	Status bit that indicates source fault handling is pending.
<code>APP_PORT_VBUS_DROP_WAIT_ACTIVE</code>	Status bit that indicates wait for VBus drop is pending.
<code>APP_PORT_DISABLE_IN_PROGRESS</code>	Port disable operation is in progress.

6.7.3.2 anonymous enum

```
anonymous enum
```

Enumerator

<code>APP_THERMISTOR_TYPE_NTC</code>	NTC Thermistor type configured.
--------------------------------------	---------------------------------

Enumerator

APP_THERMISTOR_TYPE_PTC	PTC Thermistor type configured.
APP_THERMISTOR_TYPE_ERROR	Invalid Thermistor type configured.

6.7.3.3 sys_hw_error_type_t

```
enum sys_hw_error_type_t
```

List of possible hardware errors defined for the system.

Enumerator

SYS_HW_ERROR_NONE	No error.
SYS_HW_ERROR_MUX_ACCESS	Error while accessing data MUX.
SYS_HW_ERROR_REG_ACCESS	Error while accessing regulator.
SYS_HW_ERROR_BAD_VOLTAGE	Unexpected voltage generated by source regulator.

6.7.4 Function Documentation

6.7.4.1 app_conf_for_faulty_dev_removal()

```
void app_conf_for_faulty_dev_removal (
    uint8_t port )
```

Configures TYPE-C port for fault protection state.

Parameters

port	PD port to be configured.
------	---------------------------

Returns

None

6.7.4.2 app_disable_pd_port()

```
ccg_status_t app_disable_pd_port (
    uint8_t port,
    dpm_typec_cmd_cbk_t cbk )
```

Wrapper function for PD port disable. This function is used to ensure that any application level state associated with a faulty connection are cleared when the user wants to disable a PD port.

Parameters

port	Index of port to be disabled.
cbk	Callback to be called after operation is complete.

Returns

CCG_STAT_SUCCESS on success, appropriate error code otherwise.

6.7.4.3 app_event_handler()

```
void app_event_handler (
    uint8_t port,
    app_evt_t evt,
    const void * dat )
```

Handler for event notifications from the PD stack.

Parameters

<i>port</i>	Port on which events are to be handled.
<i>evt</i>	Type of event to be handled.
<i>dat</i>	Data associated with the event.

Returns

None

6.7.4.4 app_get_callback_ptr()

```
app_cbk_t* app_get_callback_ptr (
    uint8_t port )
```

This function return the App callback structure pointer.

Parameters

<i>port</i>	port index
-------------	------------

Returns

Application callback structure pointer

6.7.4.5 app_get_resp_buf()

```
app_resp_t* app_get_resp_buf (
    uint8_t port )
```

Get a handle to the application provide PD command response buffer.

Parameters

<i>port</i>	PD port corresponding to the command and response.
-------------	--

Returns

Pointer to the response buffer.

6.7.4.6 app_get_status()

```
app_status_t* app_get_status (
    uint8_t port )
```

Get handle to structure containing information about the system status for a PD port.

Parameters

<i>port</i>	PD port to be queried.
-------------	------------------------

Returns

Pointer to the system information structure.

6.7.4.7 app_init()

```
void app_init (
    void )
```

Application level init function.

This function performs any Application level initialization required for the CCG solution. This should be called before calling the dpm_init function.

Returns

None.

6.7.4.8 app_ovp_disable()

```
void app_ovp_disable (
    uint8_t port,
    bool pfet )
```

Disable the Over-Voltage protection circuitry.

Parameters

<i>port</i>	PD port to be configured.
<i>pfet</i>	Whether PFET is used for the power supply control.

Returns

None

6.7.4.9 app_ovp_enable()

```
void app_ovp_enable (
    uint8_t port,
    uint16_t volt_mV,
    bool pfet,
    PD_ADC_CB_T ovp_cb )
```

Enable and configure the Over-Voltage protection circuitry.

Parameters

<i>port</i>	PD port to be configured.
<i>volt_mV</i>	Expected VBus voltage.
<i>pfet</i>	Whether PFET is used for the power supply control.
<i>ovp_cb</i>	Callback function to be triggered when there is an OV event.

Returns

None

6.7.4.10 app_sleep()

```
bool app_sleep (
    void )
```

Check whether the APP handlers are ready to allow device deep sleep.

Returns

true if APP handler is idle, false otherwise.

6.7.4.11 app_task()

```
uint8_t app_task (
    uint8_t port )
```

Handler for application level asynchronous tasks.

Parameters

<i>port</i>	USB-PD port for which tasks are to be handled.
-------------	--

Returns

1 in case of success, 0 in case of task handling error.

6.7.4.12 app_update_bc_src_support()

```
void app_update_bc_src_support (
```

```
    uint8_t port,
    uint8_t enable )
```

Function to update the BC 1.2 source support.

Parameters

<i>port</i>	PD port to be configured.
-------------	---------------------------

Returns

None

6.7.4.13 app_update_sys_pwr_state()

```
void app_update_sys_pwr_state (
    uint8_t state )
```

Function to update the system power state.

Parameters

<i>state</i>	Current system power state: 0=S0, None-zero=other states.
--------------	---

6.7.4.14 app_uvp_disable()

```
void app_uvp_disable (
    uint8_t port,
    bool pfet )
```

Disable the Under-Voltage protection circuitry.

Parameters

<i>port</i>	PD port to be configured.
<i>pfet</i>	Whether PFET is used for the power supply control.

Returns

None

6.7.4.15 app_uvp_enable()

```
void app_uvp_enable (
    uint8_t port,
    uint16_t volt_mV,
    bool pfet,
    PD_ADC_CB_T uvp_cb )
```

Enable and configure the Under-Voltage protection circuitry.

Parameters

<i>port</i>	PD port to be configured.
<i>volt_mV</i>	Expected VBus voltage.
<i>pfet</i>	Whether PFET is used for the power supply control.
<i>uvp_cb</i>	Callback function to be triggered when there is an UV event.

Returns

None

6.7.4.16 app_validate_configtable_offsets()

```
bool app_validate_configtable_offsets (
    void )
```

Validate the configuration table specified.

Each copy of CCGx firmware on the device flash contains an embedded configuration table that defines the runtime behaviour of the CCGx device. This function checks whether the configuration table located at the specified location is valid (has valid offsets).

Parameters

<i>null.</i>	
--------------	--

Returns

true if the table is valid, false otherwise.

6.7.4.17 app_wakeup()

```
void app_wakeup (
    void )
```

Restore the APP handler state after CCG device wakes from deep-sleep.

Returns

None

6.7.4.18 ccg_app_task_init()

```
void ccg_app_task_init (
    void )
```

Initialize CCGx periodic app level tasks.

Parameters

<i>Null</i>	<input type="button" value=""/>
-------------	---------------------------------

Returns

Null

6.7.4.19 mux_ctrl_init()

```
bool mux_ctrl_init (
    uint8_t port )
```

Initialize the Type-C Data Mux for a specific PD port.

Parameters

<i>port</i>	USB-PD port for which the MUX is to be initialized.
-------------	---

Returns

Returns true if the MUX is initialized successfully, false otherwise.

6.7.4.20 mux_ctrl_set_cfg()

```
bool mux_ctrl_set_cfg (
    uint8_t port,
    mux_select_t cfg,
    uint8_t polarity )
```

Set the Type-C MUX to the desired configuration.

Parameters

<i>port</i>	PD port on which MUX is to be configured.
<i>cfg</i>	Desired MUX configuration.
<i>polarity</i>	Polarity of the Type-C connection.

Returns

Returns true if the operation is successful, false otherwise.

6.7.4.21 sln_pd_event_handler()

```
void sln_pd_event_handler (
    uint8_t port,
    app_evt_t evt,
    const void * data )
```

Solution handler for PD events reported from the stack.

The function provides all PD events to the solution. For a solution supporting HPI, the solution function should re-direct the calls to `hpi_pd_event_handler`. If no HPI is supported, the function can be a simple dummy function.

Parameters

<i>port</i>	PD port corresponding to the event.
<i>evt</i>	Event that is being notified.
<i>data</i>	Data associated with the event. This is an opaque pointer that needs to be de-referenced based on event type.

Returns

None

6.7.4.22 system_sleep()

```
bool system_sleep (
    void )
```

Function to place CCG device in power saving mode if possible.

This function places the CCG device in power saving deep sleep mode if possible. The function checks for each interface (PD, HPI etc.) being idle and then enters sleep mode with the appropriate wake-up triggers. If the device enters sleep mode, the function will only return after the device has woken up.

Returns

true if the device went into sleep, false otherwise.

6.7.4.23 system_vconn_ocp_dis()

```
void system_vconn_ocp_dis (
    uint8_t port )
```

This function disable vconn ocp.

Parameters

<i>port</i>	Port index
-------------	------------

Returns

None

6.7.4.24 system_vconn_ocp_en()

```
bool system_vconn_ocp_en (
    uint8_t port,
    PD_ADC_CB_T cbk )
```

This function enable vconn ocp.

Parameters

<i>port</i>	Port index
<i>cbk</i>	OCP callback

Returns

Returns true on success, false if parameters are invalid.

6.7.4.25 vbus_discharge_off()

```
void vbus_discharge_off (
    uint8_t port )
```

This function turns off discharge FET on selected port.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

None

6.7.4.26 vbus_discharge_on()

```
void vbus_discharge_on (
    uint8_t port )
```

This function turns on discharge FET on selected port.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

None

6.7.4.27 vbus_get_value()

```
uint16_t vbus_get_value (
    uint8_t port )
```

This function return current VBUS voltage in mV.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

VBUS voltage in mV

6.7.4.28 vbus_is_present()

```
bool vbus_is_present (
    uint8_t port,
    uint16_t volt,
    int8_t per )
```

This function checks if power is present on VBus.

Parameters

<i>port</i>	Port index the function is performed for.
<i>volt</i>	Voltage in mV units.
<i>per</i>	Threshold margin.

Returns

true if power is present on VBus, else returns false

6.7.4.29 vconn_disable()

```
void vconn_disable (
    uint8_t port,
    uint8_t channel )
```

This function disables VCONN power.

Parameters

<i>port</i>	Port index the function is performed for.
<i>channel</i>	Selected CC line.

Returns

None

6.7.4.30 vconn_enable()

```
void vconn_enable (
```

```
    uint8_t port,
    uint8_t channel )
```

This function enables VCONN power.

Parameters

<i>port</i>	Port index the function is performed for.
<i>channel</i>	Selected CC line.

Returns

None

6.7.4.31 vconn_is_present()

```
bool vconn_is_present (
    uint8_t port )
```

This function checks if power is present on VConn.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

true if power is present on VConn, else returns false

6.8 app/billboard.h File Reference

```
#include "config.h"
#include "stdint.h"
#include "stdbool.h"
#include "status.h"
```

Data Structures

- struct [bb_handle_t](#)

Macros

- #define [BB_MAX_ALT_MODES](#) (8)
- #define [BB_ALT_MODE_STATUS_INIT_VAL](#) (0x5555)
- #define [BB_ALT_MODE_STATUS_MASK](#) (0x03)
- #define [BB_OFF_TIMER_MAX_INTERVAL](#) (1000)
- #define [BB_OFF_TIMER_NO_DISABLE](#) (0xFFFF)
- #define [BB_OFF_TIMER_MIN_VALUE](#) (60)
- #define [BB_MAX_EP0_XFER_SIZE](#) (256)
- #define [BB_CAP_ADD_FAILURE_INFO_PWR](#) (0x01)

- #define BB_CAP_ADD_FAILURE_INFO_PD (0x02)
- #define BB_DISCONNECT_STAT (0xFF)
- #define BB_CONNECT_STAT (0x01)
- #define BB_MAX_SVID (2u)

Enumerations

- enum `bb_type_t`{ BB_TYPE_NONE, BB_TYPE_EXTERNAL, BB_TYPE_INTERNAL, BB_TYPE_EXT_CONFIGURABLE }
- enum `bb_state_t`{
 BB_STATE_DEINITED, BB_STATE_DISABLED, BB_STATE_BILLBOARD, BB_STATE_LOCKED,
 BB_STATE_FLASHING }
- enum `bb_cause_t`{ BB_CAUSE_AME_TIMEOUT, BB_CAUSE_AME_SUCCESS, BB_CAUSE_AME_FAILURE,
 BB_CAUSE_PWR_FAILURE }
- enum `bb_alt_mode_status_t`{ BB_ALT_MODE_STAT_ERROR, BB_ALT_MODE_STAT_NOT_ATTEMPTED,
 BB_ALT_MODE_STAT_UNSUCCESSFUL, BB_ALT_MODE_STAT_SUCCESSFUL }
- enum `bb_usb_string_index_t`{
 BB_LANG_ID_STRING_INDEX, BB_MFG_STRING_INDEX, BB_PROD_STRING_INDEX, BB_SERIAL_STRING_INDEX,
 BB_CONFIG_STRING_INDEX, BB_BB_INF_STRING_INDEX, BB_HID_INF_STRING_INDEX, BB_URL_STRING_INDEX,
 BB_ALT_MODE1_STRING_INDEX, BB_ALT_MODE2_STRING_INDEX, BB_ALT_MODE3_STRING_INDEX,
 BB_ALT_MODE4_STRING_INDEX,
 BB_ALT_MODE5_STRING_INDEX, BB_ALT_MODE6_STRING_INDEX, BB_ALT_MODE7_STRING_INDEX,
 BB_ALT_MODE8_STRING_INDEX }

Functions

- `ccg_status_t bb_init (uint8_t port)`
- `ccg_status_t bb_enable (uint8_t port, bb_cause_t cause)`
- `ccg_status_t bb_disable (uint8_t port, bool force)`
- `ccg_status_t bb_update_alt_status (uint8_t port, uint8_t mode_index, bb_alt_mode_status_t alt_status)`
- `ccg_status_t bb_update_all_status (uint8_t port, uint32_t status)`
- `bool bb_is_present (uint8_t port)`
- `void bb_task (uint8_t port)`
- `ccg_status_t bb_flashing_ctrl (uint8_t port, bool enable)`
- `bool bb_is_idle (uint8_t port)`
- `bool bb_enter_deep_sleep (uint8_t port)`
- `ccg_status_t bb_bridge_ctrl (uint8_t port, bool enable)`
- `uint8_t * bb_get_version ()`

6.8.1 Detailed Description

Billboard control interface header file.

6.8.2 Macro Definition Documentation

6.8.2.1 BB_MAX_EP0_XFER_SIZE

```
#define BB_MAX_EP0_XFER_SIZE (256)
```

Maximum buffering for EP0 transactions inside the billboard module.

This definition is valid only for internal billboard implementation.

6.8.3 Enumeration Type Documentation

6.8.3.1 bb_alt_mode_status_t

enum `bb_alt_mode_status_t`

USB Billboard alternate mode status.

The enumeration lists the different status values as defined by the billboard class specification.

Enumerator

<code>BB_ALT_MODE_STAT_ERROR</code>	Undefined error / alternate mode does not exist
<code>BB_ALT_MODE_STAT_NOT_ATTEMPTED</code>	Alternate mode entry not attempted
<code>BB_ALT_MODE_STAT_UNSUCCESSFUL</code>	Alternate mode entry attempted but failed
<code>BB_ALT_MODE_STAT_SUCCESSFUL</code>	Alternate mode entry succeeded

6.8.3.2 bb_cause_t

enum `bb_cause_t`

USB Billboard cause for enumeration.

The enumeration lists all the supported causes for billboard enumeration.

Enumerator

<code>BB_CAUSE_AME_TIMEOUT</code>	AME timeout event
<code>BB_CAUSE_AME_SUCCESS</code>	Successful alternate mode entry
<code>BB_CAUSE_AME_FAILURE</code>	Failed alternate mode entry
<code>BB_CAUSE_PWR_FAILURE</code>	Failed to get sufficient power

6.8.3.3 bb_state_t

enum `bb_state_t`

Billboard module states.

Enumerator

<code>BB_STATE_DEINITED</code>	Module is not initialized
<code>BB_STATE_DISABLED</code>	Module is initialized but not enabled
<code>BB_STATE_BILLBOARD</code>	Module is active with billboard enumeration
<code>BB_STATE_LOCKED</code>	Module is active with additional functionality
<code>BB_STATE_FLASHING</code>	Module is active with flashing mode enumeration

6.8.3.4 bb_type_t

enum `bb_type_t`

Billboard implementation model.

Enumerator

<code>BB_TYPE_NONE</code>	No billboard device
<code>BB_TYPE_EXTERNAL</code>	External billboard device
<code>BB_TYPE_INTERNAL</code>	Device supports internal USB module to implement billboard
<code>BB_TYPE_EXT_CONFIGURABLE</code>	External billboard device which supports configuration through PD controller.

6.8.3.5 bb_usb_string_index_t

enum `bb_usb_string_index_t`

USB Billboard string descriptor indices.

The enumeration lists the different string indices used in the billboard implementation.

Enumerator

<code>BB_LANG_ID_STRING_INDEX</code>	Language ID index
<code>BB_MFG_STRING_INDEX</code>	Manufacturer string index
<code>BB_PROD_STRING_INDEX</code>	Product string index
<code>BB_SERIAL_STRING_INDEX</code>	Serial string index
<code>BB_CONFIG_STRING_INDEX</code>	Configuration string index
<code>BB_BB_INF_STRING_INDEX</code>	Billboard interface string index
<code>BB_HID_INF_STRING_INDEX</code>	HID interface string index
<code>BB_URL_STRING_INDEX</code>	Additional info URL string index
<code>BB_ALT_MODE1_STRING_INDEX</code>	Alternate mode 1 string index
<code>BB_ALT_MODE2_STRING_INDEX</code>	Alternate mode 2 string index
<code>BB_ALT_MODE3_STRING_INDEX</code>	Alternate mode 3 string index
<code>BB_ALT_MODE4_STRING_INDEX</code>	Alternate mode 4 string index
<code>BB_ALT_MODE5_STRING_INDEX</code>	Alternate mode 5 string index
<code>BB_ALT_MODE6_STRING_INDEX</code>	Alternate mode 6 string index
<code>BB_ALT_MODE7_STRING_INDEX</code>	Alternate mode 7 string index
<code>BB_ALT_MODE8_STRING_INDEX</code>	Alternate mode 8 string index

6.8.4 Function Documentation

6.8.4.1 bb_bridge_ctrl()

```
ccg_status_t bb_bridge_ctrl (
    uint8_t port,
    bool enable )
```

Function controls the bridge mode internal to the billboard module.

The function is valid only for the internal billboard implementation supporting the bridge mode extension. The function should only be called from the solution module from inside the [usb_i2cm_bridge_ctrl\(\)](#) function. This call causes re-enumeration.

Parameters

<i>port</i>	Index of the port. Caller is expected to ensure that only valid port index is provided.
-------------	---

6.8.4.2 bb_disable()

```
ccg_status_t bb_disable (
    uint8_t port,
    bool force )
```

The function queues a billboard interface disable.

The API disables the billboard device and disconnects the terminations. For internal implementation of billboard, the USB module is controlled from the [bb_task\(\)](#) and this function only queues the request. It should be noted that only one pending request is honoured. If more than one request is queued, only the latest is handled. A disable call clears any pending enable.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
<i>force</i>	Whether to force a disable. false = Interface not disabled when in flashing mode. true = Interface disabled regardless of the operation mode.

Returns

Status of the call.

6.8.4.3 bb_enable()

```
ccg_status_t bb_enable (
    uint8_t port,
    bb_cause_t cause )
```

The function queues a billboard enumeration / re-enumeration.

The API queues the billboard enumeration as per the configuration information provided. Enumeration details are retrieved from the configuration table.

The function can be called multiple times applications to trigger a re-enumeration without explicit disable call.

For internal implementation of billboard, the USB module is controlled from the [bb_task\(\)](#) and this function only queues the request. It should be noted that only one pending request is honoured. If more than one request is queued, only the latest is handled. This request is failed if the flashing mode is in progress.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
<i>cause</i>	Cause for billboard enumeration.

Returns

Status of the call.

6.8.4.4 bb_enter_deep_sleep()

```
bool bb_enter_deep_sleep (
    uint8_t port )
```

Function puts the module into deep sleep.

The function first checks if the deep sleep mode can be supported at this time and enables the module to wakeup from deep sleep. In this case, the wakeup source is USB. Deep sleep is allowed only when the USB bus is in suspend state. So it is better to check for the USB state first before this call. Failure on this call does not require any special action to be taken by the caller other than not to enter the deep sleep mode until a subsequent call passes successfully.

Parameters

<i>port</i>	Index of the port. Caller is expected to ensure that only valid port index is provided.
-------------	---

Returns

true = Successful deep sleep entry, false = Failed deep sleep entry.

6.8.4.5 bb_flashing_ctrl()

```
ccg_status_t bb_flashing_ctrl (
    uint8_t port,
    bool enable )
```

Function requests for flashing mode operation.

The function enables / disables the billboard interface in USB flashing mode. This function. This should be invoked only if the solution requires to start the flashing mode. This request shall prevent the billboard expiry on enable and re-start the billboard expiry on disable as required.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
<i>enable</i>	Enable / disable control. true = Enable, false = Disable.

Returns

Status of the call.

6.8.4.6 bb_get_version()

```
uint8_t* bb_get_version ( )
```

Function that returns the Billboard firmware version information.

Returns

Pointer to buffer containing billboard firmware version.

6.8.4.7 bb_init()

```
ccg_status_t bb_init (
    uint8_t port )
```

Initialize the billboard module.

The API initializes the billboard module. This is mainly a software state machine initialization. The USB device is not enabled at this point. The API helps to cleanup previous state information.

Parameters

<i>port</i>	Port index of port to initialize. Caller is expected to ensure that only valid port index is provided.
-------------	--

Returns

Status of the call.

6.8.4.8 bb_is_idle()

```
bool bb_is_idle (
    uint8_t port )
```

Function returns whether the billboard module is idle or not.

This function indicates whether there are any pending tasks or not. This function can be invoked before device enters sleep mode to check if it is allowed. But idle condition does not allow deep sleep entry. For this, the application is expected to use the [bb_enter_deep_sleep\(\)](#) function.

Parameters

<i>port</i>	Index of the port. Caller is expected to ensure that only valid port index is provided.
-------------	---

Returns

true = Billboard module is idle, false = billboard module is busy.

6.8.4.9 bb_is_present()

```
bool bb_is_present (
    uint8_t port )
```

Checks whether a billboard interface is present.

The function checks the configuration information and identifies if a billboard device exists.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
-------------	---

Returns

Returns true if billboard is present and false if absent.

6.8.4.10 bb_task()

```
void bb_task (
    uint8_t port )
```

Billboard module task function.

The function implements the billboard state machine and needs to be invoked in the main task loop. The task handler allows deferring interrupts and improves interrupt latency of the system.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
-------------	---

Returns

None

6.8.4.11 bb_update_all_status()

```
ccg_status_t bb_update_all_status (
    uint8_t port,
    uint32_t status )
```

The function updates the alternate mode status for all modes.

The current billboard implementation supports a maximum of 8 alternate modes and each mode as defined in the order of BOS descriptor has two bit status. Bit 1:0 indicates status of alt_mode0, Bit 3:2 indicates status of alt_mode1 etc. This function should be only used when the billboard status needs to be re-initialized to a specific value. In individual entry / exit cases, [bb_update_alt_status\(\)](#) should be used.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
<i>status</i>	Status data for all alternate modes.

Returns

Status of the call.

6.8.4.12 bb_update_alt_status()

```
ccg_status_t bb_update_alt_status (
    uint8_t port,
    uint8_t mode_index,
    bb_alt_mode_status_t alt_status )
```

The function updates the alternate mode status.

The function updates the alternate mode status information for the specified alternate mode index. The alternate mode index should match the DISCOVER_SVID and DISCOVER_MODES response for the device.

Parameters

<i>port</i>	Index of port. Caller is expected to ensure that only valid port index is provided.
<i>mode_index</i>	Index of the mode as defined by the alternate mode manager.
<i>alt_status</i>	Current alternate mode status.

Returns

Status of the call.

6.9 app/pdo.h File Reference

```
#include <pd.h>
```

Functions

- void eval_src_cap (uint8_t port, const pd_packet_t *src_cap, app_resp_cbk_t app_resp_handler)
- void eval_rdo (uint8_t port, pd_do_t rdo, app_resp_cbk_t app_resp_handler)

6.9.1 Detailed Description

PDO evaluation and handler definitions.

6.9.2 Function Documentation

6.9.2.1 eval_rdo()

```
void eval_rdo (
    uint8_t port,
    pd_do_t rdo,
    app_resp_cbk_t app_resp_handler )
```

This function can be used to ask EC to evaluate a request message For now evaluating here and executing the callback in this function itself This function can be changed as per customer design.

Parameters

<i>port</i>	Port index the function is performed for.
<i>rdo</i>	Pointer to pd packet which contains request data object.
<i>app_resp_handler</i>	Application handler callback function.

Returns

None

6.9.2.2 eval_src_cap()

```
void eval_src_cap (
    uint8_t port,
    const pd_packet_t * src_cap,
    app_resp_cbk_t app_resp_handler )
```

This function can be used to ask EC to evaluate a src cap message For now evaluating here and executing the callback in this function itself This function can be changed as per customer design.

Parameters

<i>port</i>	Port index the function is performed for.
<i>src_cap</i>	Pointer to pd packet which contains source capability.
<i>app_resp_handler</i>	Application handler callback function.

Returns

None

6.10 app/psink.h File Reference

```
#include <pd.h>
```

Functions

- void [psnk_set_voltage](#) (uint8_t port, uint16_t volt_mV)
- void [psnk_set_current](#) (uint8_t port, uint16_t cur_10mA)
- void [psnk_enable](#) (uint8_t port)
- void [psnk_disable](#) (uint8_t port, [sink_discharge_off_cbk_t](#) snk_discharge_off_handler)

6.10.1 Detailed Description

Power Sink (Consumer) manager header file.

6.10.2 Function Documentation**6.10.2.1 psnk_disable()**

```
void psnk_disable (
    uint8_t port,
    sink\_discharge\_off\_cbk\_t snk_discharge_off_handler )
```

Set VBus FET Off.

Parameters

<i>port</i>	Port index the function is performed for.
<i>snk_discharge_off_handler</i>	Sink Discharge fet off callback pointer

Returns

None

6.10.2.2 psnk_enable()

```
void psnk_enable (
    uint8_t port )
```

Set VBus FET On if device policy manager is enabled.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

None

6.10.2.3 psnk_set_current()

```
void psnk_set_current (
    uint8_t port,
    uint16_t cur_10mA )
```

This is empty function.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cur_10mA</i>	Current value in 10mA

Returns

None

6.10.2.4 psnk_set_voltage()

```
void psnk_set_voltage (
    uint8_t port,
    uint16_t volt_mV )
```

This function configures the ADC block as a comparator.

Parameters

<i>port</i>	Port index the function is performed for.
<i>volt_mV</i>	Comparator voltage level in mV

Returns

None

6.11 app/psource.h File Reference

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

Macros

- #define **PPS_CF_VBUS_DECREMENT_STEP** (200u)

Functions

- void **psrc_set_voltage** (uint8_t port, uint16_t volt_mV)
- uint32_t **psrc_get_voltage** (uint8_t port)
- void **psrc_set_current** (uint8_t port, uint16_t cur_10mA)
- void **psrc_enable** (uint8_t port, **pwr_ready_cbk_t** pwr_ready_handler)
- void **psrc_disable** (uint8_t port, **pwr_ready_cbk_t** pwr_ready_handler)

6.11.1 Detailed Description

Power source (Provider) manager header file.

6.11.2 Function Documentation

6.11.2.1 **psrc_disable()**

```
void psrc_disable (
    uint8_t port,
    pwr_ready_cbk_t pwr_ready_handler )
```

This function disables VBus power.

Parameters

<i>port</i>	Port index the function is performed for.
<i>pwr_ready_handler</i>	Application handler callback function.

Returns

None

6.11.2.2 psrc_enable()

```
void psrc_enable (
    uint8_t port,
    pwr_ready_cbk_t pwr_ready_handler )
```

This function enables VBus power.

Parameters

<i>port</i>	Port index the function is performed for.
<i>pwr_ready_handler</i>	Application handler callback function.

Returns

None

6.11.2.3 psrc_get_voltage()

```
uint32_t psrc_get_voltage (
    uint8_t port )
```

This function get Vbus voltage.

Parameters

<i>port</i>	Port index the function is performed for.
-------------	---

Returns

Voltage in 50mV units.

6.11.2.4 psrc_set_current()

```
void psrc_set_current (
    uint8_t port,
    uint16_t cur_10mA )
```

This function set VBus current.

Parameters

<i>port</i>	Port index the function is performed for.
<i>cur_10mA</i>	Current in 10mA units.

Returns

None

6.11.2.5 psrc_set_voltage()

```
void psrc_set_voltage (
    uint8_t port,
    uint16_t volt_mV )
```

This function set VBus voltage.

Parameters

<i>port</i>	Port index the function is performed for.
<i>volt_mV</i>	Voltage in mV units.

Returns

None

6.12 app/ridge_slave/ridge_slave.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "config.h"
#include "status.h"
#include "i2c.h"
```

Macros

- #define RIDGE_SLAVE_SCB_INDEX (0x1)
- #define RIDGE_SLAVE_SCB_CLOCK_FREQ (I2C_SCB_CLOCK_FREQ_1_MHZ)
- #define RIDGE_SLAVE_MIN_WRITE_SIZE (2)
- #define RIDGE_SLAVE_MAX_WRITE_SIZE (16)
- #define RIDGE_SLAVE_MAX_READ_SIZE (16)
- #define RIDGE_SLAVE_ADDR_P0 (0x38)
- #define RIDGE_SLAVE_ADDR_P1 (0x3F)
- #define RIDGE_SLAVE_ADDR_MASK (0xF0)
- #define RIDGE_CMD_CCG_RESET (0x02)
- #define RIDGE_CMD_INT_CLEAR (0x04)
- #define RIDGE_IRQ_ACK (0x2000)

Enumerations

- enum ridge_slave_reg_addr_t { RIDGE_REG_CCG_COMMAND = 0x50, RIDGE_REG_CCG_STATUS = 0x5F }

Functions

- void `ridge_slave_init` (uint8_t ar_scbnum, uint8_t portsel)
- void `ridge_slave_status_update` (uint8_t port, uint32_t status, bool rewrite)
- void `ridge_slave_task` (void)
- bool `ridge_slave_sleep` (void)
- void `ridge_reg_reset` (uint8_t port)
- bool `ridge_slave_is_host_connected` (uint8_t port)
- void `ridge_slave_update_ocp_status` (uint8_t port, bool status)

6.12.1 Detailed Description

Alpine/Titan Ridge I2C slave interface header file.

6.12.2 Macro Definition Documentation

6.12.2.1 RIDGE_SLAVE_ADDR_MASK

```
#define RIDGE_SLAVE_ADDR_MASK (0xF0)
```

I2C slave address mask to be applied on the incoming slave address.

Since a single I2C block is used to handle transaction on two different slave addresses, a mask needs to be applied while checking the incoming preamble for an address match. This mask allows comparison of only the address bits that do not change across the two addresses.

6.12.3 Enumeration Type Documentation

6.12.3.1 ridge_slave_reg_addr_t

```
enum ridge_slave_reg_addr_t
```

List of Alpine/Titan Ridge slave interface registers.

The Thunderbolt Alternate Mode specification defines the following set of registers that should be implemented by a USB-PD port controller in Thunderbolt enabled systems.

Enumerator

RIDGE_REG_CCG_COMMAND	CCG command register.
RIDGE_REG_CCG_STATUS	CCG status register.

6.12.4 Function Documentation

6.12.4.1 ridge_reg_reset()

```
void ridge_reg_reset (
    uint8_t port )
```

Resets Ridge related registers.

Parameters

<i>port</i>	USB-PD port index corresponding to the register reset.
-------------	--

Returns

None

6.12.4.2 ridge_slave_init()

```
void ridge_slave_init (
    uint8_t ar_scbnum,
    uint8_t portsel )
```

Initialize the Alpine/Titan Ridge slave interface module.

This function initializes the Alpine/Titan Ridge slave interface module and configures it to use the specified S_←CB block. The SCB will be configured as an I2C slave block, and the interrupt output will also be initialized to a de-asserted state.

Since only two registers are to be implemented, and the commands to be implemented are simple, the complete module is implemented using the I2C command callbacks.

Parameters

<i>ar_scbnum</i>	SCB index to be used for the Alpine/Titan Ridge slave interface.
<i>portsel</i>	Alpine/Titan Ridge slave port selection. Only applicable for CCG3 designs.

Returns

None

6.12.4.3 ridge_slave_is_host_connected()

```
bool ridge_slave_is_host_connected (
    uint8_t port )
```

Check whether a host is connected to the specified PD port.

Parameters

<i>port</i>	Index of the PD port.
-------------	-----------------------

Returns

true if a host is connected, false otherwise.

6.12.4.4 ridge_slave_sleep()

```
bool ridge_slave_sleep (
    void )
```

Check whether the AR/TR slave interface is idle so that device can be placed into sleep.

This function should be called prior to placing the CCG device in deep sleep. Deep sleep entry is only allowed if this function returns true.

Returns

true if the interface is idle, false otherwise.

6.12.4.5 ridge_slave_status_update()

```
void ridge_slave_status_update (
    uint8_t port,
    uint32_t status,
    bool rewrite )
```

Update the AR/TR status register and send an event to the Alpine/Titan Ridge.

This function is used by the application layer to update the content of the Alpine/Titan Ridge status register. If the content of the register is changing, CCG asserts the corresponding interrupt to notify Alpine/Titan Ridge about the status change.

Parameters

<i>port</i>	USB-PD port index corresponding to the status update.
<i>status</i>	Value to be written into the status register.
<i>rewrite</i>	Flag to enable updating of the status register even it remains the same. This feature are using for HPD status updating.

Returns

None

6.12.4.6 ridge_slave_task()

```
void ridge_slave_task (
    void )
```

Handler for pending Ridge Slave interface tasks.

Returns

None

6.12.4.7 ridge_slave_update_ocp_status()

```
void ridge_slave_update_ocp_status (
    uint8_t port,
    bool status )
```

Update the status register with Over-Current status of the port.

Parameters

<i>port</i>	PD port index.
<i>status</i>	Whether the port currently has an over-current condition.

Returns

None

6.13 app/swap.h File Reference

```
#include <config.h>
#include <pd.h>
```

Functions

- void [eval_dr_swap](#) (uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- void [eval_pr_swap](#) (uint8_t port, [app_resp_cbk_t](#) app_resp_handler)
- void [eval_vconn_swap](#) (uint8_t port, [app_resp_cbk_t](#) app_resp_handler)

6.13.1 Detailed Description

Swap request (PR_SWAP, DR_SWAP, VCONN_SWAP) handlers.

6.13.2 Function Documentation

6.13.2.1 eval_dr_swap()

```
void eval_dr_swap (
    uint8_t port,
    app\_resp\_cbk\_t app_resp_handler )
```

This function evaluates Data role swap request.

Parameters

<i>port</i>	Port index the function is performed for.
<i>app_resp_handler</i>	Application handler callback function.

Returns

None

6.13.2.2 eval_pr_swap()

```
void eval_pr_swap (
    uint8_t port,
    app_resp_cbk_t app_resp_handler )
```

This function evaluates Power role swap request.

Parameters

<i>port</i>	Port index the function is performed for.
<i>app_resp_handler</i>	Application handler callback function.

Returns

None

6.13.2.3 eval_vconn_swap()

```
void eval_vconn_swap (
    uint8_t port,
    app_resp_cbk_t app_resp_handler )
```

This function evaluates VConn swap request.

Parameters

<i>port</i>	Port index the function is performed for.
<i>app_resp_handler</i>	Application handler callback function.

Returns

None

6.14 app/usb_hid.h File Reference

```
#include "stdint.h"
#include "stdbool.h"
#include "status.h"
#include "usb.h"
```

Macros

- #define **FW_VERSION_SIZE** (8)
- #define **HID_CY_FW_INFO_SIZE** (63)

- #define HID_CUSTOMER_INFO_SIZE (32)
- #define HID_GET_SET_REPORT_ID_MASK (0xFF)
- #define USB_HID_INTR_EP_INDEX (USB_EP_INDEX_EP1)

Enumerations

- enum hid_report_id_t {
 HID_REPORT_ID_CY_FW_INFO = 0xE0, HID_REPORT_ID_RQT, HID_REPORT_ID_FLASH_WRITE,
 HID_REPORT_ID_FLASH_READ,
 HID_REPORT_ID_CUSTOMER_INFO }
- enum hid_rqt_cmd_t {
 HID_RQT_CMD_RESERVED = 0, HID_RQT_CMD_JUMP, HID_RQT_CMD_ENTER_FLASHING,
 HID_RQT_CMD_SET_READ_ROW,
 HID_RQT_CMD_VALIDATE_FW, HID_RQT_CMD_SET_APP_PRIORITY, HID_RQT_CMD_I2C_BRIDGE_CTRL
}

Functions

- uint16_t usb_hid_get_inf_dscr (uint8_t *buffer, uint16_t max_length, uint8_t inf_num, uint8_t inf_str_index)
- uint16_t usb_hid_get_report_dscr (uint8_t *buffer, uint16_t max_length)
- ccg_status_t usb_hid_class_rqt_handler (usb_setup_pkt_t *pkt)
- ccg_status_t usb_hid_inf_ctrl (bool enable)
- uint8_t * usb_hid_get_ep0_buffer (void)
- ccg_status_t usb_hid_flashing_rqt (bool enable)
- ccg_status_t usb_i2cm_bridge_ctrl (bool enable)
- void usb_hid_set_fw_locations (uint16_t bl_last_row, uint16_t fw1_last_row)

6.14.1 Detailed Description

USB HID class interface (flashing) header file.

6.14.2 Enumeration Type Documentation

6.14.2.1 hid_report_id_t

enum hid_report_id_t

HID vendor request / response report IDs.

The reports are aligned to accomodate the report ID as the first byte. The report size does not include this first byte added as part of the protocol. The byte information for each report ID includes this first byte.

Enumerator

HID_REPORT_ID_CY_FW_INFO	CY_FW_INFO data report. The report returns information about the device and firmware. Report direction: IN, report size: 63. BYTE[0] : 0xE0 BYTE[1] : Reserved BYTE[3:2] :Signature "CY" BYTE[4] : Current operating mode. BIT(1:0) - 0 = Bootloader 1 = FW image 1 2 = FW image 2 BYTE[5] : Bootloader information. BIT(0) - This bit is set if the boot-loader supports security (SHA2 checksum at boot). BIT(1) - This bit is set if the boot-loader has no flashing interface. BIT(2) - This bit is set if the boot-loader supports application priority feature. BIT(4:3) - Flash row size information 0 = Row size of 128 bytes 1 = Row size of 256 bytes BYTE[6] : Boot mode reason BIT(0) - This bit is set if the firmware requested a jump to boot-loader BIT(1) - Reserved BIT(2) - FW image 1 status. Set if invalid. BIT(3) - FW image 2 status. Set if invalid. BIT(5:4) - Application priority setting 0 = Default priority - most recent image. 1 = Image1 higher priority. 2 = Image2 higher priority. BYTE[7] : Reserved BYTE[11:8] : Silicon ID BYTE[19:12]: Bootloader version BYTE[27:20]: FW image 1 version BYTE[35:28]: FW image 2 version BYTE[39:36]: FW image 1 start address BYTE[43:40]: FW image 2 start address BYTE[51:44]: Device UID BYTE[63:52]: Reserved
HID_REPORT_ID_RQT	HID vendor command report. Report direction: OUT, report size: 7. BYTE[0] : 0xE1 BYTE[1] : Request CMD BYTE[7:2] : Command parameters.
HID_REPORT_ID_FLASH_WRITE	Flash write command report. Report direction: OUT, report size: 131. BYTE[0] : 0xE2 BYTE[1] : "F" BYTE[3:2] : Row ID to write data to. BYTE[131:4]: Data to write.
HID_REPORT_ID_FLASH_READ	Flash read command report. Report direction: IN, report size: 131. BYTE[0] : 0xE3 BYTE[1] : "F" BYTE[3:2] : Row ID of the data. BYTE[131:4]: Data read from flash.
HID_REPORT_ID_CUSTOMER_INFO	Customer information data report. Report direction: IN, report size: 32. BYTE[0] : 0xE4 BYTE[32:1] : Customer information data.

6.14.2.2 hid_rqt_cmd_t

```
enum hid_rqt_cmd_t
```

HID vendor request commands for HID_REPORT_ID_RQT_CMD report ID.

Enumerator

HID_RQT_CMD_RESERVED	Reserved command ID.
HID_RQT_CMD_JUMP	Jump request. PARAM[0] : Signature 'J' = Jump to boot-loader. 'R' = Reset device. 'A' = Jump to alternate image. PARAM[5:1]: Reserved.
HID_RQT_CMD_ENTER_FLASHING	Enter flashing mode request. PARAM[0] : Signature 'P' = Enable flashing mode. Others = Disable flashing mode. PARAM[5:1]: Reserved.
HID_RQT_CMD_SET_READ_ROW	Set flash read row request. PARAM[1:0]: Row ID PARAM[5:2]: Reserved.
HID_RQT_CMD_VALIDATE_FW	Validate firmware request. PARAM[0] : Firmware image number to validate. PARAM[5:1]: Reserved.
HID_RQT_CMD_SET_APP_PRIORITY	Set application priority setting. PARAM[0] : Signature 'F' PARAM[1] : Priority setting (0, 1 or 2). PARAM[5:2]: Reserved.

Enumerator

HID_RQT_CMD_I2C_BRIDGE_CTRL	Control the I2C master bridge mode. PARAM[0] : Signature 'B' = Enable bridge mode, else disable bridge mode. PARAM[5:1]: Reserved.
-----------------------------	--

6.14.3 Function Documentation

6.14.3.1 usb_hid_class_rqt_handler()

```
ccg_status_t usb_hid_class_rqt_handler (
    usb_setup_pkt_t * pkt )
```

USB HID-class request handler for USB flashing requests.

The function provides the default HID-class command handler for flashing. This function is expected to be invoked when a USB vendor command is received from the USB module. If there are no additional vendor commands supported by the application, then this function can be directly registered for handling vendor commands with USB module.

Parameters

<i>pkt</i>	USB setup packet received for the request
------------	---

Returns

Status of the call

6.14.3.2 usb_hid_flashing_rqt()

```
ccg_status_t usb_hid_flashing_rqt (
    bool enable )
```

Enter flashing request handler for the solution.

The function is expected to be implemented by the application. This allows the application to control the device behaviour as well as verify if flashing can be done at the specified time. If a failure is returned, the flashing request from the host shall be rejected. The expectation is that while in flashing the application does not use the USB module for any other purpose and the USB interface is left ON.

Returns

Status of the call

6.14.3.3 usb_hid_get_ep0_buffer()

```
uint8_t* usb_hid_get_ep0_buffer (
    void )
```

Get a buffer for USB EP0 request handling.

The function is expected to be implemented by the application. This allows a common buffer to be re-used for all EP0 transactions. It is expected that the buffer is always available through out the USB vendor command transaction.

The function should always return a valid pointer with sufficient buffering for a single flash row.

Returns

Pointer to the buffer

6.14.3.4 `usb_hid_get_inf_dscr()`

```
uint16_t usb_hid_get_inf_dscr (
    uint8_t * buffer,
    uint16_t max_length,
    uint8_t inf_num,
    uint8_t inf_str_index )
```

Returns the HID interface and endpoint descriptors.

The function is expected to be invoked by the global descriptor request handler and returns the HID interface descriptor and the corresponding endpoint descriptor. This can be placed directly into the configuration descriptor.

Parameters

<i>buffer</i>	Buffer pointer to copy the descriptor into. It is the responsibility of the caller to ensure that sufficient buffering is available. The HID interface descriptor along with the endpoint descriptor(s) is always expected to be less than 64 bytes.
<i>max_length</i>	Maximum buffering available
<i>inf_num</i>	Interface number to be used for creating the HID interface descriptor.
<i>inf_str_index</i>	String descriptor index to be used when creating the interface descriptor.

Returns

Actual size of the interface descriptor copied.

6.14.3.5 `usb_hid_get_report_dscr()`

```
uint16_t usb_hid_get_report_dscr (
    uint8_t * buffer,
    uint16_t max_length )
```

Returns the HID report descriptor.

The function is expected to be invoked by the global descriptor request handler and returns the HID report descriptor.

Parameters

<i>buffer</i>	Buffer pointer to copy the descriptor into. It is the responsibility of the caller to ensure that sufficient buffering is available.
<i>max_length</i>	Maximum buffering available

Returns

Actual size of the descriptor copied.

6.14.3.6 usb_hid_inf_ctrl()

```
ccg_status_t usb_hid_inf_ctrl (
    bool enable )
```

USB HID interface control function.

The function enables / disables the corresponding interface. The function is expected to be invoked from the global event handler during SET_CONFIGURATION and DISCONNECT / RESET events. The function enables / disables the interrupt endpoint for the HID interface.

Parameters

<i>enable</i>	Indicates whether to enable or disable the HID interface.
---------------	---

Returns

Status of the call

6.14.3.7 usb_hid_set_fw_locations()

```
void usb_hid_set_fw_locations (
    uint16_t bl_last_row,
    uint16_t fw1_last_row )
```

Update the firmware locations reported in the HID reports. This function can be used by solution layer code to update the default firmware locations reported by the USB-HID based firmware update code.

Parameters

<i>bl_last_row</i>	Last flash row which is part of the boot-loader.
<i>fw1_last_row</i>	Last flash row which is part of the FW1 binary.

Returns

None

6.14.3.8 usb_i2cm_bridge_ctrl()

```
ccg_status_t usb_i2cm_bridge_ctrl (
    bool enable )
```

Enter I2C master bridge mode for the solution.

The function is expected to be implemented by the application. This allows the application to enable / disable the bridge mode enumeration. If a failure is returned, the bridge mode request from the host shall be rejected.

Parameters

<code>enable</code>	True to enable and false to disable.
---------------------	--------------------------------------

Returns

Status of the call

6.15 app/usb_i2cm.h File Reference

```
#include "stdint.h"
#include "stdbool.h"
#include "status.h"
#include "usb.h"
```

Data Structures

- struct `usb_i2cm_t`

Macros

- #define `USB_I2CM_OUT_EP_INDEX` (`USB_EP_INDEX_EP2`)
- #define `USB_I2CM_IN_EP_INDEX` (`USB_EP_INDEX_EP3`)
- #define `USB_I2CM_INT_EP_INDEX` (`USB_EP_INDEX_EP4`)
- #define `USB_I2CM_SUB_CLASS` (0x03)
- #define `USB_I2CM_SCB_INDEX` (3)
- #define `USB_I2CM_CTRL_STOP_POS` (0)
- #define `USB_I2CM_CTRL_STOP` (1 << `USB_I2CM_CTRL_STOP_POS`)
- #define `USB_I2CM_CTRL_NAK_LAST_POS` (1)
- #define `USB_I2CM_CTRL_NAK_LAST` (1 << `USB_I2CM_CTRL_NAK_LAST_POS`)
- #define `USB_I2CM_CTRL_FLAGS` (0xFF)
- #define `USB_I2CM_STAT_ACTIVE_POS` (0)
- #define `USB_I2CM_STAT_ACTIVE` (1 << `USB_I2CM_STAT_ACTIVE_POS`)
- #define `USB_I2CM_STAT_BUS_BUSY_POS` (1)
- #define `USB_I2CM_STAT_BUS_BUSY` (1 << `USB_I2CM_STAT_BUS_BUSY_POS`)
- #define `USB_I2CM_STAT_TX_UFLOW_POS` (3)
- #define `USB_I2CM_STAT_TX_UFLOW` (1 << `USB_I2CM_STAT_TX_UFLOW_POS`)
- #define `USB_I2CM_STAT_ARB_ERR_POS` (4)
- #define `USB_I2CM_STAT_ARB_ERR` (1 << `USB_I2CM_STAT_ARB_ERR_POS`)
- #define `USB_I2CM_STAT_NAK_POS` (5)
- #define `USB_I2CM_STAT_NAK` (1 << `USB_I2CM_STAT_NAK_POS`)
- #define `USB_I2CM_STAT_BUS_ERR_POS` (6)
- #define `USB_I2CM_STAT_BUS_ERR` (1 << `USB_I2CM_STAT_BUS_ERR_POS`)
- #define `USB_I2CM_STAT_SIZE` (3)
- #define `USB_I2CM_BUF_SIZE` (64)

Enumerations

- enum `usb_i2cm_vdr_rqst_t`{ `USB_I2CM_VDR_RQT_I2C_WRITE` = 0xC6, `USB_I2CM_VDR_RQT_I2C_READ`, `USB_I2CM_VDR_RQT_I2C_GET_STATUS`, `USB_I2CM_VDR_RQT_I2C_RESET` }
- enum `usb_i2cm_state_t`{
 `USB_I2CM_STATE_DISABLED` = 0, `USB_I2CM_STATE_IDLE`, `USB_I2CM_STATE_WRITE`, `USB_I2CM_STATE_READ`,
 `USB_I2CM_STATE_ERROR` }

Functions

- ccg_status_t **usb_i2cm_get_dscr_rq_handler** (usb_setup_pkt_t *pkt)
- ccg_status_t **usb_i2cm_vendor_rq_handler** (usb_setup_pkt_t *pkt)
- ccg_status_t **usb_i2cm_inf_ctrl** (bool enable)
- void **usb_i2cm_task** (void)
- uint8_t * **usb_i2cm_get_ep0_buffer** (void)
- bool **usb_i2cm_is_idle** (void)

6.15.1 Detailed Description

USB I2C master bridge interface header file.

6.15.2 Enumeration Type Documentation

6.15.2.1 usb_i2cm_vdr_rq_t

enum **usb_i2cm_vdr_rq_t**

Internal USB-I2CM vendor commands enumeration.

Enumerator

USB_I2CM_VDR_RQT_I2C_WRITE	I2C write request. The data is provided over the bulk out endpoint. CMD_CODE: 0x40 VALUE: bit0 - start, bit1 - stop, bit3 - start on idle, bits[14:8] - slave addr INDEX: I2C write length. LENGTH: 0. EP0 DATA: None. OUT EP DATA: I2C data to be written.
USB_I2CM_VDR_RQT_I2C_READ	I2C read request. The data is provided over the bulk in endpoint. CMD_CODE: 0x40 VALUE: bit0 - start, bit1 - stop, bit2 - NAK last byte, bit3 - start on idle, bits[14:8] - slave addr INDEX: I2C read length. LENGTH: 0. EP0 DATA: None. IN EP DATA: I2C read data from device.
USB_I2CM_VDR_RQT_I2C_GET_STATUS	I2C status request. The current status of the I2C transaction. CMD_CODE: 0xC0 VALUE: bit0 - 0: TX 1: RX INDEX: 0. LENGTH: 3. EP0 DATA: IN - byte0: bit0 - flag, bit1 - bus_state, bit2 - SDA state, bit3 - TX underflow, bit4 - arbitration err, bit5 - NAK, bit6 - bus error byte(2:1): Data count remaining.
USB_I2CM_VDR_RQT_I2C_RESET	I2C block abort / reset request. CMD_CODE: 0x40 VALUE: bit0 - 0: TX 1: RX INDEX: 0. LENGTH: 0. EP0 DATA: None.

6.15.3 Function Documentation

6.15.3.1 **usb_i2cm_get_dscr_rq_handler()**

```
ccg_status_t usb_i2cm_get_dscr_rq_handler (
    usb_setup_pkt_t * pkt )
```

Handler for the USB-I2CM bridge GET_DESCRIPTOR request handler.

The function is expected to be called by the USB module or the solution USB request handler. There should not be any explicit call to this function.

Parameters

<i>pkt</i>	USB setup packet received for the request
------------	---

Returns

Status of the call.

6.15.3.2 `usb_i2cm_get_ep0_buffer()`

```
uint8_t* usb_i2cm_get_ep0_buffer (
    void )
```

Get a buffer for USB EP0 request handling.

The function is expected to be implemented by the application. This allows a common buffer to be re-used for all EP0 transactions. It is expected that the buffer is always available through out the USB vendor command transaction.

The function should always return a valid pointer with sufficient buffering for 64 bytes.

Returns

Pointer to the buffer

6.15.3.3 `usb_i2cm_inf_ctrl()`

```
ccg_status_t usb_i2cm_inf_ctrl (
    bool enable )
```

The function enables / disables the USB-I2CM bridge mode.

The function is expected to be called by the billboard module to enable / disable the bridge functionality. The function should be called when USB host configures / un-configures the device.

Returns

Status of the call.

6.15.3.4 `usb_i2cm_is_idle()`

```
bool usb_i2cm_is_idle (
    void )
```

Return whether the USB I2CM bridge is idle or not.

Returns

Bool: True if idle and false if not.

6.15.3.5 `usb_i2cm_task()`

```
void usb_i2cm_task (
    void )
```

Task handler for the bridge module.

The function is expected to be called by the billboard module to run the bridge tasks. The function is expected to be called once enabled from the [bb_task\(\)](#) function.

6.15.3.6 `usb_i2cm_vendor_rqt_handler()`

```
ccg_status_t usb_i2cm_vendor_rqt_handler (
    usb_setup_pkt_t * pkt )
```

Handler for the USB-I2CM bridge vendor control requests.

The function is expected to be called by the USB module or the solution USB vendor request handler. There should not be any explicit call to this function.

Parameters

<i>pkt</i>	USB setup packet received for the request
------------	---

Returns

Status of the call.

6.16 app/uvdm.h File Reference

Macros

- #define **UVDM_RESPONSE_MAX_NO_OF_VDO** (0x07)
- #define **UVDM_HEADER_INDEX** (0x00)
- #define **UVDM_SIGNATURE_BYTE_OFFSET** (0x00)
- #define **UVDM_DEVICE_MODE_VDO_INDEX** (0x01)
- #define **UVDM_BOOT_LAST_ROW_VDO_INDEX** (0x02)
- #define **UVDM_BOOT_VERSION_VDO_INDEX** (0x01)
- #define **UVDM_IMG1_VERSION_VDO_INDEX** (0x03)
- #define **UVDM_IMG2_VERSION_VDO_INDEX** (0x05)
- #define **UVDM_VERSION_NUM_SIZE_BYTES** (0x08)
- #define **UVDM_GET_VERSION_U_VDM_NO_OF_VDO** (0x06)
- #define **UVDM_GET_SILICON_ID_CMD_SIZE** (0x04)
- #define **UVDM_GET_SILICON_ID_CMD_SIG** (0x53)
- #define **UVDM_FW1_START_ADDR_VDO_INDEX** (0x01)
- #define **UVDM_FW2_START_ADDR_VDO_INDEX** (0x02)
- #define **UVDM_GET_FW_START_ADDR_UVDM_NO_OF_VDO** (0x02)
- #define **UVDM_SILICON_ID_VDO_INDEX** (0x02)
- #define **UVDM_DEVICE_RESET_CMD_SIZE** (0x04)
- #define **UVDM_DEVICE_RESET_CMD_SIG** (0x52)
- #define **UVDM_JUMP_TO_BOOT_CMD_SIZE** (0x04)
- #define **UVDM_JUMP_TO_BOOT_CMD_SIG** ('J')
- #define **UVDM_JUMP_TO_ALT_FW_SIG** ('A')
- #define **UVDM_ENTER_FLASHING_MODE_CMD_SIZE** (0x04)
- #define **UVDM_ENTER_FLASHING_MODE_CMD_SIG** (0x50)

- #define **UVDM_FLASH_READ_WRITE_CMD_SIZE** (0x04)
- #define **UVDM_FLASH_READ_WRITE_CMD_SIG** (0x46)
- #define **UVDM_FLASH_ROW_NUM_LSB_OFFSET** (0x01)
- #define **UVDM_FLASH_ROW_NUM_MSB_OFFSET** (0x02)
- #define **UVDM_READ_DATA_RESPONSE_VDO_INDEX** (0x01)
- #define **UVDM_RESPONSE_VDO_INDEX** (0x01)
- #define **UVDM_READ_DATA_NO_OF_VDO_ERROR_CASE** (0x02)
- #define **UVDM_VALIDATE_FW_CMD_SIZE** (0x04)
- #define **UVDM_VALIDATE_FW_MODE_INDEX** (0x00)
- #define **UVDM_REASON_FOR_BOOT_MODE_VDO_INDEX** (0x01)
- #define **UVDM_GET_CHECKSUM_CMD_SIZE** (0x08)
- #define **UVDM_FLASH_ADDR_LSB_OFFSET** (0x00)
- #define **UVDM_FLASH_SIZE_LSB_OFFSET** (0x04)
- #define **UVDM_CHEKCSUM_VDO_INDEX** (0x01)
- #define **UVDM_SET_APP_PRIORITY_CMD_SIZE** (0x04)
- #define **UVDM_SET_APP_PRIORITY_INDEX** (0x00)
- #define **UVDM_SEND_SIGN_SEQUENCE_1** (0x01)
- #define **UVDM_SEND_SIGN_SEQUENCE_2** (0x02)
- #define **UVDM_SEND_SIGN_SEQUENCE_3** (0x03)
- #define **UVDM_SEND_SIGN_SEC_1_2_SIZE** (0x18)
- #define **UVDM_SEND_SIGN_SEC_3_SIZE** (0x10)
- #define **UVDM_GET_CUSTOMER_INFO_SEQ_1** (0x01)
- #define **UVDM_GET_CUSTOMER_INFO_SEQ_2** (0x02)
- #define **UVDM_GET_CUSTOMER_INFO_RESPONSE_SIZE** (0x10)
- #define **UVDM_GET_CUSTOMER_INFO_RESPONSE_VDO_NUM** (0x04)

Enumerations

- enum **uvdm_cmd_opcode_t** {
 UVDM_CMD_RESERVED, UVDM_CMD_GET_DEVICE_MODE_OPCODE, UVDM_CMD_GET_DEVICE_VERSION_OPCODE,
 UVDM_CMD_GET_SILICON_ID_OPCODE,
 UVDM_CMD_DEVICE_RESET_OPCODE, UVDM_CMD_JUMP_TO_BOOT_OPCODE, UVDM_CMD_ENTER_FLASHING_MODE,
 UVDM_CMD_SEND_DATA_OPCODE,
 UVDM_CMD_FLASH_WRITE_OPCODE, UVDM_CMD_READ_DATA_OPCODE, UVDM_CMD_FLASH_READ_OPCODE,
 UVDM_CMD_VALIDATE_FW_OPCODE,
 UVDM_CMD_REASON_FOR_BOOT_MODE, UVDM_CMD_GET_CHECKSUM, UVDM_CMD_GET_FW_START_ADDRESS,
 UVDM_CMD_SET_APP_PRIORITY_OPCODE,
 UVDM_CMD_RESERVED_16, UVDM_CMD_SEND_SIGNATURE_OPCODE, UVDM_CMD_RESERVED_18,
 UVDM_CMD_GET_BOOT_TYPE,
 UVDM_CMD_GET_CUSTOMER_INFO }
- enum **uvdm_response_state_t** { **UVDM_NOT_HANDLED**, **UVDM_HANDLED_RESPONSE_READY**,
 UVDM_HANDLED_NO_RESPONSE, **UVDM_HANDLED_RESPONSE_NOT_READY** }
- enum **uvdm_qc_4_0_cmd_t** {
 UVDM_QC_GET_CASE_TEMP = 0x1003, **UVDM_QC_GET_CONNECTOR_TEMP** = 0x0B03, **UVDM_QC_GET_CONNECTOR_VOLT** = 0x0603, **UVDM_QC_GET_CHARGER_TYPE** = 0x0C03,
 UVDM_QC_GET_CHARGER_VERSION = 0xE03 }

Functions

- **uvdm_cmd_opcode_t uvdm_get_cur_nb_cmd** (void)
- **void uvdm_reset_nb_cmd_state** (void)
- **void uvdm_enter_cy_alt_mode** (void)
- **void uvdm_exit_cy_alt_mode** (void)
- **bool uvdm_get_flashing_mode** (void)

- `ccg_status_t uvd़m_handle_device_reset (uint32_t reset_sig)`
- `uvdm_response_state_t uvd़m_handle_cmd (uint32_t *rx_pkt, pd_do_t **vdm_rspn_pkt, uint8_t *vdo_count, flash_cbk_t flash_cb)`
- `uvdm_response_state_t uvd़m_qc_4_0_handler (uint8_t port, uint32_t *rx_pkt, pd_do_t **vdm_rspn_pkt, uint8_t *vdo_count)`

6.16.1 Detailed Description

Unstructured VDM handler header file.

6.16.2 Enumeration Type Documentation

6.16.2.1 uvd़m_cmd_opcode_t

```
enum uvd़m_cmd_opcode_t
```

Enumerator

UVDM_CMD_RESERVED	Reserved.
UVDM_CMD_GET_DEVICE_MODE_OPCODE	Get active mode of device.
UVDM_CMD_GET_DEVICE_VERSION_OPCODE	Get version information of all images.
UVDM_CMD_GET_SILICON_ID_OPCODE	Get silicon ID of CCG.
UVDM_CMD_DEVICE_RESET_OPCODE	Reset CCG.
UVDM_CMD_JUMP_TO_BOOT_OPCODE	Jump to boot mode.
UVDM_CMD_ENTER_FLASHING_MODE_OPCODE	Enable flash access mode.
UVDM_CMD_SEND_DATA_OPCODE	Collect flash row data for write.
UVDM_CMD_FLASH_WRITE_OPCODE	Program flash row.
UVDM_CMD_READ_DATA_OPCODE	Collect flash row data for read.
UVDM_CMD_FLASH_READ_OPCODE	Read flash row.
UVDM_CMD_VALIDATE_FW_OPCODE	Validate FW image.
UVDM_CMD_REASON_FOR_BOOT_MODE	Get reason for boot mode.
UVDM_CMD_GET_CHECKSUM	Get checksum of specified flash section.
UVDM_CMD_GET_FW_START_ADDRESS_OPCODE	Get start address of FW images.
UVDM_CMD_SET_APP_PRIORITY_OPCODE	Set APP priority value.
UVDM_CMD_RESERVED_16	Reserved.
UVDM_CMD_SEND_SIGNATURE_OPCODE	Program FW image ECDSA signature.
UVDM_CMD_RESERVED_18	Reserved.
UVDM_CMD_GET_BOOT_TYPE	Get information related to boot loader type.
UVDM_CMD_GET_CUSTOMER_INFO	Get custom data.

6.16.2.2 uvd़m_response_state_t

```
enum uvd़m_response_state_t
```

Enumerator

UVDM_NOT_HANDLED	UVDM not recognised, can't be handled.
------------------	--

Enumerator

UVDM_HANDLED_RESPONSE_READY	UVDM handled and response is ready to be sent.
UVDM_HANDLED_NO_RESPONSE	UVDM handled but no response required.
UVDM_HANDLED_RESPONSE_NOT_READY	UVDM Handled but response will be sent later, potential non-blocking command.

6.17 app/vdm.h File Reference

```
#include <pd.h>
```

Functions

- void `vdm_data_init` (uint8_t port)
- void `vdm_update_data` (uint8_t port, uint8_t id_vdo_cnt, uint8_t *id_vdo_p, uint8_t svnid_vdo_cnt, uint8_t *svnid_vdo_p, uint16_t mode_resp_len, uint8_t *mode_resp_p)
- void `eval_vdm` (uint8_t port, const `pd_packet_t` *vdm, `vdm_resp_cbk_t` vdm_resp_handler)
- bool `get_modes_vdo_info` (uint8_t port, uint16_t svnid, `pd_do_t` **temp_p, uint8_t *no_of_vdo)
- void `vdm_update_svid_resp` (uint8_t port, uint8_t svnid_vdo_cnt, uint8_t *svnid_vdo_p)

6.17.1 Detailed Description

Vendor Defined Message (VDM) handler header file.

6.17.2 Function Documentation

6.17.2.1 eval_vdm()

```
void eval_vdm (
    uint8_t port,
    const pd_packet_t * vdm,
    vdm_resp_cbk_t vdm_resp_handler )
```

This function is responsible for analysing and processing received VDM. This function also makes a decision about necessity of response to the received VDM.

Parameters

<code>port</code>	Port index the function is performed for.
<code>vdm</code>	Pointer to pd packet which contains received VDM.
<code>vdm_resp_handler</code>	VDM handler callback function.

Returns

None

6.17.2.2 get_modes_vdo_info()

```
bool get_modes_vdo_info (
    uint8_t port,
    uint16_t svid,
    pd_do_t ** temp_p,
    uint8_t * no_of_vdo )
```

This function is responsible for getting Discover Mode info from config table .

Parameters

<i>port</i>	Port index the function is performed for.
<i>svid</i>	SVID which the infomation is searching for.
<i>temp_p</i>	Temporary pointer to the pointer of PD data object.
<i>no_of_vdo</i>	Pointer to the variable which contains number of VDOs in Disc MODE response.

Returns

Returns true if config table contains Disc MODE info for input SVID. Else returns false.

6.17.2.3 vdm_data_init()

```
void vdm_data_init (
    uint8_t port )
```

Store the VDM data from the configuration table.

This function retrieves the VDM data (for CCG as UFP) that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be stored.
-------------	---

Returns

None.

6.17.2.4 vdm_update_data()

```
void vdm_update_data (
    uint8_t port,
    uint8_t id_vdo_cnt,
    uint8_t * id_vdo_p,
    uint8_t svid_vdo_cnt,
    uint8_t * svid_vdo_p,
    uint16_t mode_resp_len,
    uint8_t * mode_resp_p )
```

This function allows the VDM data for CCG to be changed.

This function allows the user to change the VDM responses that CCG sends for D_ID, D_SVID and D_MODE requests. The default responses are taken from the configuration table. This function allows the user to change

the response data. The caller is responsible to ensure that the responses are not changed while CCG is already in contract as a UFP.

Parameters

<i>port</i>	PD port for which responses are to be changed.
<i>id_vdo_cnt</i>	Number of VDOs in the D_ID response.
<i>id_vdo_p</i>	Pointer to the actual D_ID response in memory.
<i>svid_vdo_cnt</i>	Number of VDOs in the D_SVID response. Should be less than 8.
<i>svid_vdo_p</i>	Pointer to the actual D_SVID response in memory.
<i>mode_resp_len</i>	Total length of mode response. This includes the D_MODE responses for each supported SVID, along with the corresponding header fields.
<i>mode_resp_p</i>	Pointer to all of the mode responses in memory.

Returns

None

6.17.2.5 vdm_update_svid_resp()

```
void vdm_update_svid_resp (
    uint8_t port,
    uint8_t svid_vdo_cnt,
    uint8_t * svid_vdo_p )
```

Update the Discover SVID response sent by the device. This method is provided so that the list of supported SVIDs can be changed dynamically without having to change all of the VDO configuration.

Parameters

<i>port</i>	PD port to be updated.
<i>svid_vdo_cnt</i>	Number of VDOs in the D_SVID response. Should be less than 8.
<i>svid_vdo_p</i>	Pointer to the actual D_SVID response in memory.

Returns

None

6.18 hpiss/hpi.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <config.h>
#include "status.h"
#include "i2c.h"
#include "pd.h"
#include "dpm.h"
#include "app.h"
```

Macros

- #define HPI_PD_CMD_TIMER (0x80)
- #define HPI_ADDR_I2C_CFG_LOW (0x40)
- #define HPI_ADDR_I2C_CFG_HIGH (0x42)
- #define HPI_ADDR_I2C_CFG_FLOAT (0x08)

TypeDefs

- typedef uint8_t(* hpi_write_cb_t) (uint16_t reg_addr, uint8_t wr_size, uint8_t *wr_data)

Enumerations

- enum hpi_reg_section_t { HPI_REG_SECTION_DEV = 0, HPI_REG_SECTION_PORT_0, HPI_REG_SECTION_PORT_1, HPI_REG_SECTION_ALL }
- enum hpi_reg_part_t {
 HPI_REG_PART_REG = 0, HPI_REG_PART_DATA = 1, HPI_REG_PART_FLASH = 2, HPI_REG_PART_PDDATA_READ = 4,
 HPI_REG_PART_PDDATA_READ_H = 5, HPI_REG_PART_PDDATA_WRITE = 8, HPI_REG_PART_PDDATA_WRITE_H = 9 }
- enum hpi_boot_prio_conf_t { HPI_BOOT_PRIO_LAST_FLASHED = 0, HPI_BOOT_PRIO_APP_PRIO_ROW, HPI_BOOT_PRIO_FW1, HPI_BOOT_PRIO_FW2 }

Functions

- void hpi_init (uint8_t scb_idx)
- void hpi_deinit (void)
- void hpi_task (void)
- bool hpi_reg_enqueue_event (hpi_reg_section_t section, uint8_t status, uint16_t length, uint8_t *data)
- void hpi_pd_event_handler (uint8_t port, app_evt_t evt, const void *data)
- bool hpi_is_ec_ready (void)
- bool hpi_is_accessed (void)
- void hpi_update_versions (uint8_t *bl_version, uint8_t *fw1_version, uint8_t *fw2_version)
- void hpi_set_mode_regs (uint8_t dev_mode, uint8_t mode_reason)
- void hpi_update_fw_locations (uint16_t fw1_location, uint16_t fw2_location)
- bool hpi_is_vdm_ec_ctrl_enabled (uint8_t port)
- bool hpi_is_extd_msg_ec_ctrl_enabled (uint8_t port)
- uint8_t hpid_get_ec_active_modes (uint8_t port)
- bool hpi_sleep_allowed (void)
- bool hpi_sleep (void)
- uint8_t hpi_get_port_enable (void)
- void hpi_set_no_boot_mode (bool enable)
- void hpi_set_fixed_slave_address (uint8_t slave_addr)
- void hpi_send_fw_ready_event (void)
- void hpi_set_flash_params (uint32_t flash_size, uint16_t row_size, uint16_t row_cnt, uint16_t bl_last_row)
- ccg_status_t hpi_init_userdef_regs (uint16_t reg_addr, uint8_t size, uint8_t *data)
- void hpi_set_userdef_write_handler (hpi_write_cb_t wr_handler)
- void hpi_set_reset_count (uint8_t count)
- void hpi_set_reserved_reg_35 (uint16_t value)
- void hpi_set_reserved_reg_37 (uint16_t value)
- void hpi_set_port_event_mask (uint8_t port, uint32_t mask)
- void hpi_send_hw_error_event (uint8_t port, uint8_t err_type)
- uint8_t hpi_get_sys_pwr_state (void)
- void hpi_set_boot_priority_conf (uint8_t conf)

- void [hpi_update_pdo_change](#) (bool disable)
- uint16_t [get_hpi_soft_reset_delay](#) (void)
- void [update_hpi_soft_reset_delay](#) (uint16_t value)
- uint8_t [get_hpi_sof_reset_timer_id](#) (void)
- void [update_hpi_sof_reset_timer_id](#) (uint8_t value)

6.18.1 Detailed Description

Host Processor Interface (HPI) header file.

6.18.2 Typedef Documentation

6.18.2.1 `hpi_write_cb_t`

`hpi_write_cb_t`

Handler for HPI register writes.

Returns

Type of response to be sent to the EC. Only a single byte response can be sent from here. Use `hpi_reg_enqueue_event` to send longer responses.

6.18.3 Enumeration Type Documentation

6.18.3.1 `hpi_boot_prio_conf_t`

enum `hpi_boot_prio_conf_t`

Enumeration showing possible boot priority configurations for the firmware application.

Enumerator

<code>HPI_BOOT_PRIO_LAST_FLASHED</code>	Last flashed firmware is prioritized.
<code>HPI_BOOT_PRIO_APP_PRIO_ROW</code>	Priority defined used App Priority flash row.
<code>HPI_BOOT_PRIO_FW1</code>	FW1 is always prioritized.
<code>HPI_BOOT_PRIO_FW2</code>	FW2 is always prioritized.

6.18.3.2 `hpi_reg_part_t`

enum `hpi_reg_part_t`

Types of HPI register/memory regions.

Enumerator

<code>HPI_REG_PART_REG</code>	Register region.
-------------------------------	------------------

Enumerator

HPI_REG_PART_DATA	Data memory for device section.
HPI_REG_PART_FLASH	Flash memory.
HPI_REG_PART_PDDATA_READ	Read Data memory for port section.
HPI_REG_PART_PDDATA_READ_H	Upper fraction of read data memory for port section.
HPI_REG_PART_PDDATA_WRITE	Write Data memory for port section.
HPI_REG_PART_PDDATA_WRITE_H	Upper fraction of write data memory for port section.

6.18.3.3 hpi_reg_section_t

```
enum hpi_reg_section_t
```

HPI register section definitions.

HPI registers are grouped into sections corresponding to the functions that are supported.

Enumerator

HPI_REG_SECTION_DEV	Device information registers.
HPI_REG_SECTION_PORT_0	USB-PD Port 0 related registers.
HPI_REG_SECTION_PORT_1	USB-PD Port 1 related registers.
HPI_REG_SECTION_ALL	Special definition to select all register spaces.

6.18.4 Function Documentation

6.18.4.1 get_hpi_sof_reset_timer_id()

```
uint8_t get_hpi_sof_reset_timer_id (
    void )
```

Get the Timer ID related to HPI soft reset delay.

Returns

Delay Timer ID

6.18.4.2 get_hpi_soft_reset_delay()

```
uint16_t get_hpi_soft_reset_delay (
    void )
```

Get the HPI soft reset delay value.

Returns

Delay value

6.18.4.3 hpi_deinit()

```
void hpi_deinit (
    void )
```

De-initialize the HPI interface.

This function can be used to de-initialize the HPI interface. This can be used in applications where the HPI master (Billboard controller) in the system may be powered off under control of the CCG device.

Returns

None

6.18.4.4 hpi_get_port_enable()

```
uint8_t hpi_get_port_enable (
    void )
```

Get the Port Enable register value.

Returns

The Port Enable HPI register value.

6.18.4.5 hpi_get_sys_pwr_state()

```
uint8_t hpi_get_sys_pwr_state (
    void )
```

Get the content of the HPI system power state register.

Returns

The 8-bit unsigned content of the HPI syspwr_state register.

6.18.4.6 hpi_init()

```
void hpi_init (
    uint8_t scb_idx )
```

Initialize the HPI interface.

This function initializes the I2C interface and EC_INT GPIO used for HPI hardware interface, and initializes all HPI registers to their default values.

Parameters

<i>scb_idx</i>	Index of SCB block to be used for HPI. Please note that this parameter is not validated, and it is the caller's responsibility to pass the correct value.
----------------	---

Returns

None

6.18.4.7 hpi_init_userdef_regs()

```
ccg_status_t hpi_init_userdef_regs (
    uint16_t reg_addr,
    uint8_t size,
    uint8_t * data )
```

This function initializes the user-defined HPI registers.

This function is used to initialize the contents of the user-defined registers that are part of the HPI register space.

Parameters

<i>reg_addr</i>	The base address of registers to be updated. Should be in the user defined address region.
<i>size</i>	Number of registers to be updated. The upper limit should not exceed the user defined address region.
<i>data</i>	Buffer containing data to be copied into HPI registers.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.18.4.8 hpi_is_accessed()

```
bool hpi_is_accessed (
    void )
```

Check whether any HPI accesses have happened since start-up.

Returns

True if any HPI read/write access has happened.

6.18.4.9 hpi_is_ec_ready()

```
bool hpi_is_ec_ready (
    void )
```

Check whether EC init complete event has been received.

This function is used by the application to check whether the EC has sent the EC initialization complete event notification.

Returns

true if EC init has been received, false otherwise.

6.18.4.10 hpi_is_extd_msg_ec_ctrl_enabled()

```
bool hpi_is_extd_msg_ec_ctrl_enabled (
    uint8_t port )
```

Check whether handling of extended messages by EC is enabled. If not enabled, CCG firmware will automatically respond with NOT_SUPPORTED messages.

Parameters

<i>port</i>	USB-PD port to check the configuration for.
-------------	---

Returns

true if EC handling of extended messages is enabled, false otherwise.

6.18.4.11 hpi_is_vdm_ec_ctrl_enabled()

```
bool hpi_is_vdm_ec_ctrl_enabled (
    uint8_t port )
```

Check whether EC control of VDMs is enabled.

Parameters

<i>port</i>	USB-PD port to check the configuration for.
-------------	---

Returns

true if EC control is enabled, false otherwise.

6.18.4.12 hpi_pd_event_handler()

```
void hpi_pd_event_handler (
    uint8_t port,
    app_evt_t evt,
    const void * data )
```

Handler for PD events reported from the stack.

Internal function used to receive PD events from the stack and to update the HPI registers.

Parameters

<i>port</i>	PD port corresponding to the event.
<i>evt</i>	Event that is being notified.
<i>data</i>	Data associated with the event. This is an opaque pointer that needs to be de-referenced based on event type.

Returns

None

6.18.4.13 hpi_reg_enqueue_event()

```
bool hpi_reg_enqueue_event (
    hpi_reg_section_t section,
    uint8_t status,
    uint16_t length,
    uint8_t * data )
```

Enqueue an event to the EC through the HPI interface.

This function is used by the PD stack and application layers to send event notifications to the EC through the HPI registers.

Please note that only responses without associated data can be sent through the response register for the HPI_REG_SECTION_DEV section. If any additional response data is required, please use the user defined registers or the response register associated with HPI_REG_SECTION_PORT_0 or HPI_REG_SECTION_PORT_1.

Parameters

<i>section</i>	Register section through which event is to be reported.
<i>status</i>	The event code to be stored into the response register.
<i>length</i>	Length of the data associated with the event.
<i>data</i>	Pointer to buffer containing data associated with the event.

Returns

true if the event queue has space for the event, false if there is an overflow.

6.18.4.14 hpi_send_fw_ready_event()

```
void hpi_send_fw_ready_event (
    void )
```

Send a FW ready notification through HPI to the EC. This event is sent to the EC to indicate that the device is out of reset and has loaded firmware.

Returns

None

6.18.4.15 hpi_send_hw_error_event()

```
void hpi_send_hw_error_event (
    uint8_t port,
    uint8_t err_type )
```

Notify EC about a system hardware access error.

Parameters

<i>port</i>	PD port index.
<i>err_type</i>	Type of error detected.

Returns

None

6.18.4.16 hpi_set_boot_priority_conf()

```
void hpi_set_boot_priority_conf (
    uint8_t conf )
```

Update the firmware boot priority configuration reported through HPI.

Parameters

<i>conf</i>	Firmware boot priority supported by the firmware.
-------------	---

Returns

None

6.18.4.17 hpi_set_fixed_slave_address()

```
void hpi_set_fixed_slave_address (
    uint8_t slave_addr )
```

Set the I2C slave address to be used for the HPI interface.

Parameters

<i>slave_addr</i>	Slave address to be used.
-------------------	---------------------------

Returns

None

6.18.4.18 hpi_set_flash_params()

```
void hpi_set_flash_params (
    uint32_t flash_size,
    uint16_t row_size,
    uint16_t row_cnt,
    uint16_t bl_last_row )
```

Set the CCG device flash parameters. These values are used for the device status reporting and firmware update implementation.

Parameters

<i>flash_size</i>	Total device flash size in bytes.
<i>row_size</i>	Size of each flash row in bytes.
<i>row_cnt</i>	Number of flash rows on the device.
<i>bl_last_row</i>	Last flash row assigned to boot-loader.

Returns

None

6.18.4.19 hpi_set_mode_regs()

```
void hpi_set_mode_regs (
    uint8_t dev_mode,
    uint8_t mode_reason )
```

Set device mode and reason register values.

This is an internal function used to update the device mode and boot mode reason HPI registers.

Parameters

<i>dev_mode</i>	Value to be set into the device mode register.
<i>mode_reason</i>	Value to be set into the boot mode reason register.

Returns

void

6.18.4.20 hpi_set_no_boot_mode()

```
void hpi_set_no_boot_mode (
    bool enable )
```

Configure HPI to operate in No-boot support mode.

Parameters

<i>enable</i>	Whether to enable no-boot mode.
---------------	---------------------------------

Returns

None

6.18.4.21 hpi_set_port_event_mask()

```
void hpi_set_port_event_mask (
```

```
    uint8_t port,
    uint32_t mask )
```

Update the event mask value for the specified PD port.

Parameters

<i>port</i>	PD port index.
<i>mask</i>	Event mask value to be set.

Returns

None

6.18.4.22 hpi_set_reserved_reg_35()

```
void hpi_set_reserved_reg_35 (
    uint16_t value )
```

Debug API to update registers at address 0x35 - 0x34.

Parameters

<i>value</i>	16-bit value to be updated into HPI registers 0x35 and 0x34.
--------------	--

Returns

None

6.18.4.23 hpi_set_reserved_reg_37()

```
void hpi_set_reserved_reg_37 (
    uint16_t value )
```

Debug API to update registers at address 0x37 - 0x36.

Parameters

<i>value</i>	16-bit value to be updated into HPI registers 0x37 and 0x36.
--------------	--

Returns

None

6.18.4.24 hpi_set_reset_count()

```
void hpi_set_reset_count (
    uint8_t count )
```

Store the reset counter value into the appropriate HPI register.

Parameters

<i>count</i>	The reset count to be stored.
--------------	-------------------------------

Returns

None

6.18.4.25 hpi_set_userdef_write_handler()

```
void hpi_set_userdef_write_handler (
    hpi_write_cb_t wr_handler )
```

Enable handling of user-defined register writes in the Application.

This function is used to enable handling of EC writes to the user-defined HPI register region in the application code.

Parameters

<i>wr_handler</i>	Pointer to function that handles the received HPI writes.
-------------------	---

6.18.4.26 hpi_sleep()

```
bool hpi_sleep (
    void )
```

Prepare the HPI interface for device deep sleep.

This function checks whether the I2C interface is idle so that the CCG device can enter deep sleep mode. It also enables an I2C address match as a wake-up trigger from deep sleep. `hpi_sleep_allowed` should have been called prior to calling this function.

Returns

true if the HPI interface is ready for sleep, false otherwise.

6.18.4.27 hpi_sleep_allowed()

```
bool hpi_sleep_allowed (
    void )
```

Check if the CCG device can be put into deep-sleep.

Returns

true if deep sleep is possible, false otherwise.

6.18.4.28 hpi_task()

```
void hpi_task (
    void )
```

HPI task handler.

This function handles the commands from the EC through the HPI registers. HPI writes from the EC are handled in interrupt context, and any associated work is queued to be handled by this function. The hpi_task is expected to be called periodically from the main task loop of the firmware application.

Returns

None

6.18.4.29 hpi_update_fw_locations()

```
void hpi_update_fw_locations (
    uint16_t fw1_location,
    uint16_t fw2_location )
```

Update the firmware location HPI registers.

This is an internal function used to update the firmware binary location HPI registers.

Parameters

<i>fw1_location</i>	Flash row where FW1 is located.
<i>fw2_location</i>	Flash row where FW2 is located.

Returns

void

6.18.4.30 hpi_update_pdo_change()

```
void hpi_update_pdo_change (
    bool disable )
```

Enable/disable PDO update through HPI.

Parameters

<i>disable</i>	Whether PDO update is to be disabled.
----------------	---------------------------------------

Returns

None

6.18.4.31 hpi_update_versions()

```
void hpi_update_versions (
    uint8_t * bl_version,
    uint8_t * fw1_version,
    uint8_t * fw2_version )
```

Update firmware version information in HPI registers.

This is an internal function used to update the firmware version information in the HPI registers.

Parameters

<i>bl_version</i>	Buffer containing Bootloader version information.
<i>fw1_version</i>	Buffer containing firmware-1 version information.
<i>fw2_version</i>	Buffer containing firmware-2 version information.

Returns

void

6.18.4.32 hpid_get_ec_active_modes()

```
uint8_t hpid_get_ec_active_modes (
    uint8_t port )
```

Get the active EC alternate modes value.

Parameters

<i>port</i>	USB-PD to check the configuration for.
-------------	--

Returns

The Active EC modes setting programmed by EC.

6.18.4.33 update_hpi_sof_reset_timer_id()

```
void update_hpi_sof_reset_timer_id (
    uint8_t value )
```

Update the Timer ID related to HPI soft reset delay.

Parameters

<i>value</i>	Timer ID
--------------	----------

Returns

None

6.18.4.34 update_hpi_soft_reset_delay()

```
void update_hpi_soft_reset_delay (
    uint16_t value )
```

Update the HPI soft reset delay value.

Parameters

<code>value</code>	Delay value in ms
--------------------	-------------------

Returns

None

6.19 pd_common/dpm.h File Reference

```
#include "pd.h"
#include "status.h"
```

Functions

- `bool dpm_refresh_src_cap (uint8_t port)`
- `bool dpm_refresh_snk_cap (uint8_t port)`
- `ccg_status_t dpm_init (uint8_t port, app_cbk_t *app_cbk)`
- `ccg_status_t dpm_start (uint8_t port)`
- `ccg_status_t dpm_stop (uint8_t port)`
- `ccg_status_t dpm_disable (uint8_t port)`
- `bool dpm_deepsleep (void)`
- `bool dpm_wakeup (void)`
- `bool dpm_sleep (void)`
- `ccg_status_t dpm_task (uint8_t port)`
- `const dpm_status_t * dpm_get_info (uint8_t port)`
- `void dpm_update_def_cable_cap (uint16_t def_cur)`
- `uint16_t dpm_get_def_cable_cap (void)`
- `void dpm_update_snk_wait_cap_period (uint16_t period)`
- `uint16_t dpm_get_snk_wait_cap_period (void)`
- `void dpm_update_mux_enable_wait_period (uint16_t period)`
- `uint16_t dpm_get_mux_enable_wait_period (void)`
- `ccg_status_t dpm_pd_command (uint8_t port, dpm_pd_cmd_t cmd, dpm_pd_cmd_buf_t *buf_ptr, dpm_pd_cmd_cbk_t cmd_cbk)`
- `ccg_status_t dpm_pd_command_ec (uint8_t port, dpm_pd_cmd_t cmd, dpm_pd_cmd_buf_t *buf_ptr, dpm_pd_cmd_cbk_t cmd_cbk)`
- `ccg_status_t dpm_typec_command (uint8_t port, dpm_typec_cmd_t cmd, dpm_typec_cmd_cbk_t cmd_cbk)`
- `ccg_status_t dpm_update_swap_response (uint8_t port, uint8_t value)`
- `ccg_status_t dpm_update_src_cap (uint8_t port, uint8_t count, pd_do_t *pdo)`
- `ccg_status_t dpm_update_src_cap_mask (uint8_t port, uint8_t mask)`
- `ccg_status_t dpm_update_snk_cap (uint8_t port, uint8_t count, pd_do_t *pdo)`
- `ccg_status_t dpm_update_snk_cap_mask (uint8_t port, uint8_t mask)`
- `ccg_status_t dpm_update_snk_max_min (uint8_t port, uint8_t count, uint16_t *max_min)`
- `ccg_status_t dpm_update_port_config (uint8_t port, uint8_t role, uint8_t dflt_role, uint8_t toggle_en, uint8_t try_src_snk_en)`
- `ccg_status_t dpm_is_rdo_valid (uint8_t port, pd_do_t rdo)`

- `uint8_t dpm_get_polarity (uint8_t port)`
- `ccg_status_t dpm_typec_deassert_rp_rd (uint8_t port, uint8_t channel)`
- `void dpm_update_port_status (uint8_t port, uint8_t input, uint8_t battery)`
- `uint32_t dpm_get_pd_port_status (uint8_t port)`
- `void dpm_update_frs_enable (uint8_t port, bool frs_rx_en, bool frs_tx_en)`
- `void dpm_update_ext_src_cap (uint8_t port, uint8_t *buf_p)`
- `ccg_status_t dpm_prot_reset (uint8_t port, sop_t sop)`
- `ccg_status_t dpm_prot_reset_rx (uint8_t port, sop_t sop)`
- `uint8_t dpm_get_evtno_n_clear (uint8_t max_events, volatile uint32_t *evt_ptr)`
- `ccg_status_t dpm_pe_stop (uint8_t port)`
- `ccg_status_t dpm_set_alert (uint8_t port, pd_do_t alert_ado)`
- `ccg_status_t dpm_set_cf (uint8_t port, bool status)`
- `ccg_status_t dpm_clear_hard_reset_count (uint8_t port)`
- `ccg_status_t dpm_set_fault_active (uint8_t port)`
- `ccg_status_t dpm_clear_fault_active (uint8_t port)`
- `ccg_status_t dpm_set_chunk_transfer_running (uint8_t port, pd_ams_type ams_type)`
- `ccg_status_t dpm_send_hard_reset (uint8_t port, uint8_t reason)`
- `int8_t dpm_get_sink_detach_margin (uint8_t port)`
- `uint16_t dpm_get_sink_detach_voltage (uint8_t port)`
- `void dpm_pps_task (uint8_t port)`

Variables

- `port_type_t gl_dpm_port_type [NO_OF_TYPEC_PORTS]`

6.19.1 Detailed Description

Device Policy Manager (DPM) header file.

6.19.2 Function Documentation

6.19.2.1 dpm_clear_fault_active()

```
ccg_status_t dpm_clear_fault_active (
    uint8_t port )
```

This function clears internal fault flag.

Parameters

<code>port</code>	Port index
-------------------	------------

Returns

`ccg_status_t`

6.19.2.2 dpm_clear_hard_reset_count()

```
ccg_status_t dpm_clear_hard_reset_count (
```

```
    uint8_t port )
```

This function clears hard reset count. This is specifically provided for VBUS fault handlers.

Parameters

<i>port</i>	Port index
-------------	------------

Returns

```
ccg_status_t
```

6.19.2.3 dpm_deepsleep()

```
bool dpm_deepsleep (  
    void )
```

This function configures the device policy manager, for all ports, so that the system can go to deepsleep. This function does not put the system into deepsleep but only performs the necessary tasks to enter deepsleep, if entry is possible.

Returns

Returns true if deepsleep is possible and configured, false otherwise.

6.19.2.4 dpm_disable()

```
ccg_status_t dpm_disable (  
    uint8_t port )
```

This function disables PD port operation and limits it to receiving hard reset signaling.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.5 dpm_get_def_cable_cap()

```
uint16_t dpm_get_def_cable_cap (  
    void )
```

This function returns the default cable current characteristics for the stack. The parameter is used by the stack to determine the maximum current setting allowed when no e-marked cable is present.

Returns

Default current setting in 10mA unit (3A is represented as 300).

6.19.2.6 dpm_get_evtno_n_clear()

```
uint8_t dpm_get_evtno_n_clear (
    uint8_t max_events,
    volatile uint32_t * evt_ptr )
```

This function retrieve the event number from event bitmask and also clears the original event variable. PE and Type C state machines use bitmask event variable. Maximum no of events possible. Pointer to the event variable.

Returns

Returns event number

Warning

This function should not be called if event is equal to zero. This is an internal function. Not required for normal usecase.

6.19.2.7 dpm_get_info()

```
const dpm_status_t* dpm_get_info (
    uint8_t port )
```

This function returns device policy manager status information for the specified port.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

Pointer to a structure containing the DPM status information.

Warning

The information provided by this API must not be altered by the application.

6.19.2.8 dpm_get_mux_enable_wait_period()

```
uint16_t dpm_get_mux_enable_wait_period (
    void )
```

Returns the current delay between the MUX enable and VBus turn ON.

Returns

Current delay in ms.

6.19.2.9 dpm_get_pd_port_status()

```
uint32_t dpm_get_pd_port_status (
    uint8_t port )
```

Get the PD port status.

Parameters

<i>port</i>	PD port to be queried.
-------------	------------------------

Returns

32-bit PD port status value to be reported through HPI.

6.19.2.10 dpm_get_polarity()

```
uint8_t dpm_get_polarity (
    uint8_t port )
```

Get the CC polarity of the Type-C connection.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

Returns 0 if CC1 is used, and 1 if CC2 is used.

6.19.2.11 dpm_get_sink_detach_margin()

```
int8_t dpm_get_sink_detach_margin (
    uint8_t port )
```

This function returns the margin on the VBUS below which sink can assume detach has happened.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

% margin

6.19.2.12 dpm_get_sink_detach_voltage()

```
uint16_t dpm_get_sink_detach_voltage (
    uint8_t port )
```

This function returns the nominal VBUS voltage threshold for which sink will detect a disconnect. Note: Sink margin will also applied on this nominal value to determine actual threshold.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

VBUS voltage in mV units

6.19.2.13 dpm_get_snk_wait_cap_period()

```
uint16_t dpm_get_snk_wait_cap_period (
    void )
```

Returns the current tTypeCSnkWaitCap value used by the CCG PD stack. This function is used by the stack to get the default or user selected timer period.

Returns

Current tTypeCSnkWaitCap timer period.

6.19.2.14 dpm_init()

```
ccg_status_t dpm_init (
    uint8_t port,
    app_cbk_t * app_cbk )
```

This function initializes the device policy manager with callback pointers and loads the system info from config table. This function also initializes policy engine and type c manager. This function must be called once on system init.

Parameters

<i>port</i>	Port index.
<i>app_cbk</i>	Application callback function pointer.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.15 dpm_is_rdo_valid()

```
ccg_status_t dpm_is_rdo_valid (
    uint8_t port,
    pd_do_t rdo )
```

This generic function is provided by the device policy manager to evaluate any RDO with respect to current source cap of the specified port.

Parameters

<i>port</i>	Port index.
<i>rdo</i>	Request data object.

Returns

CCG_STAT_SUCCESS if rdo is valid, CCG_STAT_BAD_PARAM if port index is not correct, CCG_STAT_FAILURE if rdo is not valid.

6.19.2.16 dpm_pd_command()

```
ccg_status_t dpm_pd_command (
    uint8_t port,
    dpm_pd_cmd_t cmd,
    dpm_pd_cmd_buf_t * buf_ptr,
    dpm_pd_cmd_cbk_t cmd_cbk )
```

This function provides an interface for the application module to send PD commands.

Parameters

<i>port</i>	Port index.
<i>cmd</i>	Command name.
<i>buf_ptr</i>	Pointer to the command buffer.
<i>cmd_cbk</i>	Pointer to the callback function.

Returns

Returns CCG_STAT_SUCCESS if the command is registered, CCG_STAT_CMD_FAILURE if the PD port is not ready for a command and CCG_STAT_BUSY if there is another pending command.

Warning

Data received via the callback should be copied out by the application, because the buffer will be reused by the stack to store data on new message reception.

6.19.2.17 dpm_pd_command_ec()

```
ccg_status_t dpm_pd_command_ec (
    uint8_t port,
    dpm_pd_cmd_t cmd,
    dpm_pd_cmd_buf_t * buf_ptr,
    dpm_pd_cmd_cbk_t cmd_cbk )
```

This function provides an interface for the HPI module to send PD commands. This is only meant for HPI module wherein responses come from EC.

Parameters

<i>port</i>	Port index.
-------------	-------------

Parameters

<i>cmd</i>	Command name.
<i>buf_ptr</i>	Pointer to the command buffer.
<i>cmd_cbk</i>	Pointer to the callback function.

Returns

Returns CCG_STAT_SUCCESS if the command is registered, CCG_STAT_CMD_FAILURE if the PD port is not ready for a command and CCG_STAT_BUSY if there is another pending command.

Warning

Data received via the callback should be copied out by the application, because the buffer will be reused by the stack to store data on new message reception.

6.19.2.18 dpm_pe_stop()

```
ccg_status_t dpm_pe_stop (
    uint8_t port )
```

This function stops the policy engine. Used in fault scenario wherein PD protocol need to be stopped but type c manager still runs.

Parameters

<i>port</i>	port index
-------------	------------

Returns

ccg_status_t

6.19.2.19 dpm_prot_reset()

```
ccg_status_t dpm_prot_reset (
    uint8_t port,
    sop_t sop )
```

This function resets protocol layer(RX and TX) counter for a specific sop type.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type

Returns

ccg_status_t

6.19.2.20 dpm_prot_reset_rx()

```
ccg_status_t dpm_prot_reset_rx (
    uint8_t port,
    sop_t sop )
```

This function resets protocol layer RX only counter for a specific sop type.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type

Returns

ccg_status_t

6.19.2.21 dpm_send_hard_reset()

```
ccg_status_t dpm_send_hard_reset (
    uint8_t port,
    uint8_t reason )
```

Try to send a HardReset to the port partner.

Parameters

<i>port</i>	PD port index.
<i>reason</i>	Reason for the hard reset.

Returns

ccg_status_t

6.19.2.22 dpm_set_alert()

```
ccg_status_t dpm_set_alert (
    uint8_t port,
    pd_do_t alert_ado )
```

This function sets alert ADO on ocp/ovp fault. Stack will automatically send alert after explicit contract when alert ADO is non zero. Once alert is sent or detach happens alert ADO will be cleared automatically by stack.

Parameters

<i>port</i>	port index
<i>alert_ado</i>	Alert Augmented Data Object.

Returns

```
ccg_status_t
```

6.19.2.23 dpm_set_cf()

```
ccg_status_t dpm_set_cf (
    uint8_t port,
    bool status )
```

This function sets/clears current foldback status. When in current foldback mode this function can be used to convey the status to stack.

Parameters

<i>port</i>	Port index
<i>status</i>	CF status

Returns

```
ccg_status_t
```

6.19.2.24 dpm_set_chunk_transfer_running()

```
ccg_status_t dpm_set_chunk_transfer_running (
    uint8_t port,
    pd_ams_type ams_type )
```

This function sets chunk transfer type. This should be called after response of a chunk is received to inform stack that chunked transfer is not over yet.

Parameters

<i>port</i>	Port index
-------------	------------

Returns

```
ccg_status_t
```

6.19.2.25 dpm_set_fault_active()

```
ccg_status_t dpm_set_fault_active (
    uint8_t port )
```

This function set internal flag if any fault is active ocp/ovp/otp etc.

Parameters

<i>port</i>	Port index
-------------	------------

Returns`ccg_status_t`**6.19.2.26 dpm_sleep()**

```
bool dpm_sleep (
    void )
```

This function checks if the device policy manager can go into sleep mode.

Returns

Returns true if possible to go into sleep mode, false otherwise.

6.19.2.27 dpm_start()

```
ccg_status_t dpm_start (
    uint8_t port )
```

This function makes the specified port operational.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM if port index is not correct or dpm_init is not done, CCG_STAT_FAILURE if port is disabled.

6.19.2.28 dpm_stop()

```
ccg_status_t dpm_stop (
    uint8_t port )
```

This function stops the port operation. The port will be put into the lowest power state and will not be operational.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.29 dpm_task()

```
ccg_status_t dpm_task (
    uint8_t port )
```

This function runs the device policy manager task for the specified port.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.30 dpm_typec_command()

```
ccg_status_t dpm_typec_command (
    uint8_t port,
    dpm_typec_cmd_t cmd,
    dpm_typec_cmd_cbk_t cmd_cbk )
```

This function provides an interface for the application module to control the Type C interface.

Parameters

<i>port</i>	Port index.
<i>cmd</i>	Command name.
<i>cmd_cbk</i>	Pointer to the callback function.

Returns

Returns CCG_STAT_SUCCESS if the command is registered, CCG_STAT_BUSY if a previous command is still active and CCG_STAT_CMD_FAILURE if the port is in a disabled state.

6.19.2.31 dpm_typec_deassert_rp_rd()

```
ccg_status_t dpm_typec_deassert_rp_rd (
    uint8_t port,
    uint8_t channel )
```

This function removes Rp and Rd from the specified CC channel.

Parameters

<i>port</i>	Port index.
<i>channel</i>	CC channel.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.32 dpm_update_def_cable_cap()

```
void dpm_update_def_cable_cap (
    uint16_t def_cur )
```

This function updates the default cable current characteristics. The parameter is used by the stack to determine the maximum current setting allowed when no e-marked cable is present. The default spec limit is 3A. It should be overridden only for specific captive cable solution with a different current capability. The function should be called only once before the DPM is initialized and the parameter is used for all ports on the device. If the function is not called, the default value of 3A is used.

Parameters

<i>def_cur</i>	Default current setting in 10mA unit (3A is represented as 300).
----------------	--

Warning

The function should be called only when for a non-standard solution and has the current capability.

6.19.2.33 dpm_update_ext_src_cap()

```
void dpm_update_ext_src_cap (
    uint8_t port,
    uint8_t * buf_p )
```

Update the extended source capabilities for the PD port.

Parameters

<i>port</i>	PD port to be updated.
<i>buf_p</i>	Pointer to buffer containing extended source capabilities data.

Returns

None

6.19.2.34 dpm_update_frs_enable()

```
void dpm_update_frs_enable (
    uint8_t port,
    bool frs_rx_en,
    bool frs_tx_en )
```

Enable/disable the PD 3.0 FRS functionality.

Parameters

<i>port</i>	PD port to be updated.
-------------	------------------------

Parameters

<i>frs_rx_en</i>	Whether FRS receive is to be enabled.
<i>frs_tx_en</i>	Whether FRS transmit is to be enabled.

Returns

None

6.19.2.35 dpm_update_mux_enable_wait_period()

```
void dpm_update_mux_enable_wait_period (
    uint16_t period )
```

Function to specify the delay to be used between the APP_EVT_TYPEC_ATTACH which is used to enable the Data Mux and the turning ON of the power source. By default, the stack does not use any delay. In cases where the MUX used requires time to properly turn ON, this function can be used to delay the VBus turn ON.

Parameters

<i>period</i>	Delay to be provided (in ms) between MUX enable and VBus turning ON.
---------------	--

Returns

None

6.19.2.36 dpm_update_port_config()

```
ccg_status_t dpm_update_port_config (
    uint8_t port,
    uint8_t role,
    uint8_t dfilt_role,
    uint8_t toggle_en,
    uint8_t try_src_snk_en )
```

Change the PD port configuration at runtime.

This function allows changing the PD port configuration parameters like port role, default port role, DRP toggle enable and Try.Src enable at runtime. These changes are only allowed while the corresponding PD port is disabled.

Parameters

<i>port</i>	USB-PD port to be configured.
<i>role</i>	New port role selection (0 = Sink, 1 = Source, 2 = Dual Role).
<i>dfilt_role</i>	New default port role selection (0 = Sink, 1 = Source).
<i>toggle_en</i>	New value for DRP toggle enable flag.
<i>try_src_snk_en</i>	New value for Try.SRC/ TRY.SNK enable flag(0 = Both Try.SRC and TRY.SNK are disabled, 1 = Try.SRC is enabled, 2 = TRY.SNK is enabled).

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM if port index is not correct or other parameters are not correct, CCG_STAT_FAILURE if port is not disabled.

6.19.2.37 dpm_update_port_status()

```
void dpm_update_port_status (
    uint8_t port,
    uint8_t input,
    uint8_t battery )
```

Updates the PD port status.

Parameters

<i>port</i>	PD port to be updated.
<i>input</i>	Present input status.
<i>battery</i>	Present battery status.

Returns

void

6.19.2.38 dpm_update_snk_cap()

```
ccg_status_t dpm_update_snk_cap (
    uint8_t port,
    uint8_t count,
    pd_do_t * pdo )
```

This function updates the sink PDOs at runtime thereby overriding the sink PDOs in the config table.

Parameters

<i>port</i>	Port index.
<i>count</i>	Count of PDOs.
<i>pdo</i>	Pointer to the PDO array.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.39 dpm_update_snk_cap_mask()

```
ccg_status_t dpm_update_snk_cap_mask (
    uint8_t port,
    uint8_t mask )
```

This function updates the sink PDO mask at runtime thereby overriding the sink PDO mask specified in the config table.

Parameters

<i>port</i>	Port index.
<i>mask</i>	PDO mask.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.40 dpm_update_snk_max_min()

```
ccg_status_t dpm_update_snk_max_min (
    uint8_t port,
    uint8_t count,
    uint16_t * max_min )
```

This function updates the sink max/min current/power at runtime thereby overriding the sink max/min current/power specified in the config table.

Parameters

<i>port</i>	Port index.
<i>count</i>	Count of PDOs.
<i>max_min</i>	Pointer to max/min cur/power array.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.41 dpm_update_snk_wait_cap_period()

```
void dpm_update_snk_wait_cap_period (
    uint16_t period )
```

Function to update the tTypeCSnkWaitCap period used by the CCG PD stack. The default tTypeCSnkWaitCap value used by CCG PD stack is 400 ms which is the mid-value of the spec defined range. However, this timer only starts when the firmware has started running; which could be about 250 ms from device power up in some cases. This function allows the user to update the tTypeCSnkWaitCap period based on the worst case start-up delays for the application.

Parameters

<i>period</i>	The tTypeCSnkWaitCap period to be used, in ms.
---------------	--

Returns

None

6.19.2.42 dpm_update_src_cap()

```
ccg_status_t dpm_update_src_cap (
    uint8_t port,
    uint8_t count,
    pd_do_t * pdo )
```

This function updates the source PDOs at runtime thereby overriding the source PDOs in the config table.

Parameters

<i>port</i>	Port index.
<i>count</i>	Count of PDOs.
<i>pdo</i>	Pointer to the PDO array.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.43 dpm_update_src_cap_mask()

```
ccg_status_t dpm_update_src_cap_mask (
    uint8_t port,
    uint8_t mask )
```

This function updates the source PDO mask at runtime thereby overriding the source PDO mask in the config table.

Parameters

<i>port</i>	Port index.
<i>mask</i>	PDO mask.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.44 dpm_update_swap_response()

```
ccg_status_t dpm_update_swap_response (
    uint8_t port,
    uint8_t value )
```

Update the SWAP_RESPONSE setting for the device policy manager.

Parameters

<i>port</i>	Port index.
<i>value</i>	New value for swap response.

Returns

CCG_STAT_SUCCESS if operation is successful, CCG_STAT_BAD_PARAM otherwise.

6.19.2.45 dpm_wakeup()

```
bool dpm_wakeup (
    void )
```

This function configures the device policy manager, for all ports, after the system comes out of deepsleep.

Returns

Returns true if successful, 0 otherwise.

6.20 pd_common/pd.h File Reference

```
#include <config.h>
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <utils.h>
```

Data Structures

- union `cc_state_t`
- union `pd_hdr_t`
- struct `pd_hdr_t::PD_HDR`
- union `pd_extd_hdr_t`
- union `pd_do_t`
- struct `pd_do_t::BIST_DO`
- struct `pd_do_t::FIXED_SRC`
- struct `pd_do_t::VAR_SRC`
- struct `pd_do_t::BAT_SRC`
- struct `pd_do_t::SRC_GEN`
- struct `pd_do_t::FIXED_SNK`
- struct `pd_do_t::VAR_SNK`
- struct `pd_do_t::BAT_SNK`
- struct `pd_do_t::RDO_FIXED_VAR`
- struct `pd_do_t::RDO_FIXED_VAR_GIVEBACK`
- struct `pd_do_t::RDO_BAT`
- struct `pd_do_t::RDO_BAT_GIVEBACK`
- struct `pd_do_t::RDO_GEN`
- struct `pd_do_t::RDO_GEN_GVB`
- struct `pd_do_t::STD_VDM_HDR`

- struct `pd_do_t::USTD_VDM_HDR`
- struct `pd_do_t::USTD_QC_4_0_HDR`
- struct `pd_do_t::QC_4_0_DATA_VDO`
- struct `pd_do_t::STD_VDM_ID_HDR`
- struct `pd_do_t::STD_CERT_VDO`
- struct `pd_do_t::STD_PROD_VDO`
- struct `pd_do_t::STD_CBL_VDO`
- struct `pd_do_t::PAS_CBL_VDO`
- struct `pd_do_t::ACT_CBL_VDO`
- struct `pd_do_t::STD_AMA_VDO`
- struct `pd_do_t::STD_AMA_VDO_PD3`
- struct `pd_do_t::STD_SVID_RESP_VDO`
- struct `pd_do_t::STD_DP_VDO`
- struct `pd_do_t::DP_STATUS_VDO`
- struct `pd_do_t::DP_CONFIG_VDO`
- struct `pd_power_status_t`
- union `pd_port_status_t`
- struct `pd_port_status_t::PD_PORT_STAT`
- struct `pd_port_config_t`
- struct `ovp_settings_t`
- struct `ocp_settings_t`
- struct `uvp_settings_t`
- struct `scp_settings_t`
- struct `vconn_ocp_settings_t`
- struct `otp_settings_t`
- struct `pwr_params_t`
- struct `chg_cfg_params_t`
- struct `bat_chg_params_t`
- struct `bb_settings_t`
- struct `pd_config_t`
- struct `app_resp_t`
- struct `vdm_resp_t`
- struct `dpm_pd_cmd_buf_t`
- struct `contract_t`
- struct `pd_contract_info_t`
- struct `pd_packet_t`
- struct `pd_packet_extd_t`
- struct `app_cbk_t`
- struct `dpm_status_t`

Macros

- `#define PD_EXTERNALLY_POWERED_BIT_POS (7u)`
- `#define BC_SINK_1_2_MODE_ENABLE_MASK (0x01)`
- `#define BC_SINK_APPLE_MODE_ENABLE_MASK (0x02)`
- `#define BC_SRC_1_2_MODE_ENABLE_MASK (0x01)`
- `#define BC_SRC_APPLE_MODE_ENABLE_MASK (0x02)`
- `#define BC_SRC_QC_MODE_ENABLE_MASK (0x04)`
- `#define BC_SRC_AFC_MODE_ENABLE_MASK (0x08)`
- `#define BC_SRC_QC_4_0_MODE_ENABLE_MASK (0x10)`
- `#define BC_SRC_QC_VER_2_CLASS_A_VAL (0u)`
- `#define BC_SRC_QC_VER_2_CLASS_B_VAL (1u)`
- `#define BC_SRC_QC_VER_3_CLASS_A_VAL (2u)`
- `#define BC_SRC_QC_VER_3_CLASS_B_VAL (3u)`

- #define PD_MAX_SRC_CAP_TRY (6u)
- #define GIVE_BACK_MASK (0x8000u)
- #define SNK_MIN_MAX_MASK (0x3FFu)
- #define GET_DR_SWAP_RESP(resp) ((resp) & 0x3u)
- #define GET_PR_SWAP_RESP(resp) (((resp) & 0xCu) >> 2u)
- #define GET_VCONN_SWAP_RESP(resp) (((resp) & 0x30u) >> 4u)
- #define MAX_SRC_CAP_COUNT (50u)
- #define MAX_HARD_RESET_COUNT (3u)
- #define MAX_CBL_DSC_ID_COUNT (20u)
- #define MAX_NO_OF_DO (7u)
- #define MAX_NO_OF_PDO (MAX_NO_OF_DO)
- #define MAX_NO_OF_VDO (MAX_NO_OF_DO)
- #define VDM_HEADER_IDX (0u)
- #define BDO_HDR_IDX (0u)
- #define ID_HEADER_IDX (1u)
- #define RDO_IDX (0u)
- #define MAX_EXTD_PKT_SIZE (260u)
- #define MAX_EXTD_PKT_WORDS (65u)
- #define MAX_EXTD_MSG_LEGACY_LEN (26u)
- #define MAX_MESSAGE_ID (7u)
- #define MAX_SOP_TYPES (3)
- #define STD_SVID (0xFF00u)
- #define DP_SVID (0xFF01u)
- #define TBT_SVID (0x8087u)
- #define CY_VID (0x04B4u)
- #define STD_VDM_VERSION_IDX (13u)
- #define STD_VDM_VERSION_REV3 (1u)
- #define STD_VDM_VERSION_REV2 (0u)
- #define STD_VDM_VERSION (0u)
- #define VSAFE_0V (800u)
- #define VSAFE_3_6V (3600u)
- #define VSAFE_5V (5000u)
- #define VSAFE_9V (9000u)
- #define VSAFE_12V (12000u)
- #define VSAFE_13V (13000u)
- #define VSAFE_15V (15000u)
- #define VSAFE_19V (19000u)
- #define VSAFE_20V (20000u)
- #define VSAFE_0V_PR_SWAP_SNK_SRC (3000u)
- #define VSAFE_0V_HARD_RESET (3000u)
- #define PD_VOLT_PER_UNIT (50u)
- #define ISAFE_0A (0u)
- #define ISAFE_DEF (50u)
- #define I_1A (100)
- #define I_1P5A (150)
- #define I_2A (200)
- #define I_3A (300)
- #define I_5A (500)
- #define PD_CUR_PER_UNIT (10u)
- #define SNK_DETACH_VBUS_POLL_COUNT (5u)
- #define DRP_TOGGLE_PERIOD (75u)
- #define SRC_DRP_MIN_DC (30)
- #define CC_CHANNEL_1 (0u)
- #define CC_CHANNEL_2 (1u)
- #define TYPEC_FSM_NONE (0x00000000u)

- #define TYPEC_FSM_GENERIC (0x00000001u)
- #define HPD_RX_ACTIVITY_TIMER_PERIOD_MIN (5u)
- #define HPD_RX_ACTIVITY_TIMER_PERIOD_MAX (105u)
- #define PD_NO_RESPONSE_TIMER_PERIOD (5000u)
- #define PD_CBL_DSC_ID_TIMER_PERIOD (49u)
- #define PD_CBL_DSC_ID_START_TIMER_PERIOD (43u)
- #define PD_CBL_DELAY_TIMER_PERIOD (2u)
- #define PD_PHY_BUSY_TIMER_PERIOD (15u)
- #define PD_HARD_RESET_TX_TIMER_PERIOD (20u)
- #define PD_VCONN_SWAP_INITIATOR_TIMER_PERIOD (110u)
- #define PD_VBUS_TURN_ON_TIMER_PERIOD (275u)
- #define PD_VBUS_TURN_OFF_TIMER_PERIOD (625u)
- #define PD_PS_SRC_TRANS_TIMER_PERIOD (400u)
- #define PD_PS_SRC_OFF_TIMER_PERIOD (900u)
- #define PD_PS_SRC_ON_TIMER_PERIOD (450u)
- #define PD_PS_SNK_TRANSITION_TIMER_PERIOD (500u)
- #define PD_SRC_RECOVER_TIMER_PERIOD (800u)
- #define PD_SENDER_RESPONSE_TIMER_PERIOD (27u)
- #define PD_RECEIVER_RESPONSE_TIMER_PERIOD (15u)
- #define PD_SINK_WAIT_CAP_TIMER_PERIOD (400u)
- #define PD_SRC_CAP_TIMER_PERIOD (150u)
- #define PD_SWAP_SRC_START_TIMER_PERIOD (55u)
- #define PD_SOURCE_TRANSITION_TIMER_PERIOD (28u)
- #define PD_VCONN_OFF_TIMER_PERIOD (25u)
- #define PD_VCONN_ON_TIMER_PERIOD (100u)
- #define PD_VCONN_TURN_ON_TIMER_PERIOD (10u)
- #define PD_CBL_READY_TIMER_PERIOD (50u)
- #define PD_VDM_RESPONSE_TIMER_PERIOD (25u)
- #define PD_VDM_ENTER_MODE_RESPONSE_TIMER_PERIOD (45u)
- #define PD_VDM_EXIT_MODE_RESPONSE_TIMER_PERIOD (45u)
- #define PD_DPM_RESP_REC_RESP_PERIOD (20u)
- #define PD_BIST_CONT_MODE_TIMER_PERIOD (55u)
- #define PD_SINK_VBUS_TURN_OFF_TIMER_PERIOD (750u)
- #define PD_SINK_VBUS_TURN_ON_TIMER_PERIOD (1300u)
- #define PD_PS_HARD_RESET_TIMER_PERIOD (27u)
- #define PD_COLLISION_SRC_COOL_OFF_TIMER_PERIOD (5u)
- #define PD_SINK_TX_TIMER_PERIOD (18u)
- #define PD_PPS_SRC_TIMER_PERIOD (14000u)
- #define TYPEC_CC_DEBOUNCE_TIMER_PERIOD (140u)
- #define TYPEC_PD_DEBOUNCE_TIMER_PERIOD (11u)
- #define TYPEC_RD_DEBOUNCE_TIMER_PERIOD (12u)
- #define TYPEC_SRC_DETACH_DEBOUNCE_PERIOD (2u)
- #define TYPEC_PD3_RPCHANGE_DEBOUNCE_PERIOD (2u)
- #define TYPEC_ERROR_RECOVERY_TIMER_PERIOD (50u)
- #define TYPEC_DRP_TRY_TIMER_PERIOD (110u)
- #define CCG_FRS_TX_ENABLE_MASK (0x02u)
- #define CCG_FRS_RX_ENABLE_MASK (0x01u)
- #define CCG_PD_EXT_SRCCAP_SIZE (24u)
- #define CCG_PD_EXT_SRCCAP_INP_INDEX (21u)
- #define CCG_PD_EXT_SRCCAP_INP_UNCONSTRAINED (0x02u)
- #define CCG_PD_EXT_SRCCAP_PDP_INDEX (23u)
- #define CCG_PD_EXT_STATUS_SIZE (5u)
- #define CCG_PD_EXT_PPS_STATUS_SIZE (4u)
- #define CCG_PD_FIX_SRC_PDO_MASK_REV2 (0xFE3FFFFFFu)
- #define CCG_PD_FIX_SRC_PDO_MASK_REV3 (0xFF3FFFFFFu)

- #define CCG_PD_FLAG_CONTRACT_NEG_ACTIVE (1u)
- #define CCG_PD_FLAG_EXPLICIT_CONTRACT (2u)
- #define CCG_PD_FLAG_SRC_READY (4u)
- #define CCG_PD_FLAG_POWER_SINK (8u)
- #define CCG_CC_STAT_ZOPEN (0u)
- #define CCG_CC_STAT_DRP_TOGGLE (1u)
- #define CCG_CC_STAT_RD_PRESENT (2u)
- #define CCG_CC_STAT_RP_PRESENT (4u)
- #define CCG_CC_STAT_VCONN_ACTIVE (8u)
- #define CCG_DPM_ERROR_NONE (0u)
- #define CCG_DPM_ERROR_NO_VCONN (1u)

Typedefs

- typedef struct `pwr_params_t` **typeA_pwr_params_t**
- typedef struct `chg_cfg_params_t` **typeA_chg_cfg_params_t**
- typedef void(* `pd_cbk_t`) (uint8_t port, uint32_t event)
- typedef void(* `dpm_pd_cmd_cbk_t`) (uint8_t port, `resp_status_t` resp, const `pd_packet_t` *pkt_ptr)
- typedef void(* `app_resp_cbk_t`) (uint8_t port, `app_resp_t` *resp)
- typedef void(* `vdm_resp_cbk_t`) (uint8_t port, `vdm_resp_t` *resp)
- typedef void(* `pwr_ready_cbk_t`) (uint8_t port)
- typedef void(* `sink_discharge_off_cbk_t`) (uint8_t port)
- typedef void(* `dpm_typec_cmd_cbk_t`) (uint8_t port, `dpm_typec_cmd_resp_t` resp)

Enumerations

- enum `pd_err_recov_reason_t` {
 `ERR_RECOV_REASON_NONE` = 0, `ERR_RECOV_HR_FAIL`, `ERR_RECOV_PROTECT_FAULT`,
 `ERR_RECOV_POWER_FAULT`,
 `ERR_RECOV_BAD_DATA_ROLE`, `ERR_RECOV_FRS_FAIL` }
- enum `pd_emca_sr_reason_t` { `EMCA_SR_REASON_NONE` = 0, `EMCA_SR_CABLE_DISC`, `EMCA_SR_ALT_MODE_DISC` }
- enum `pd_cable_reset_reason_t` { `EMCA_CABLE_RES_NONE` = 0, `EMCA_CABLE_RES_SR_TIMEOUT` }
- enum `pd_hard_reset_reason_t` {
 `PD_HARDRES_REASON_NONE` = 0, `PD_HARDRES_REASON_NO_SRC_CAP`, `PD_HARDRES_REASON_HOSTCONN`,
 `PD_HARDRES_REASON_SR_ERROR`,
 `PD_HARDRES_REASON_CONTRACT_ERROR`, `PD_HARDRES_REASON_DRSWAP`, `PD_HARDRES_REASON_VBUS_O`,
 `PD_HARDRES_REASON_VBUS_OCP`,
 `PD_HARDRES_REASON_AMS_ERROR` }
- enum `pd_soft_reset_reason_t` { `PD_SOFTRES_REASON_NONE` = 0, `PD_SOFTRES_REASON_SRCNEG_ERROR`,
 `PD_SOFTRES_REASON_SNKNEG_ERROR`, `PD_SOFTRES_REASON_AMS_ERROR` }
- enum `pd_timer_id_t` {
 `PD_TIMERS_START_ID` = 0, `PD_CABLE_TIMER` = 1u, `PD_NO_RESPONSE_TIMER` = 2u, `PD_CBL_DSC_ID_TIMER`
 = 3u,
 `PD_CBL_DELAY_TIMER` = 4u, `PD_PHY_BUSY_TIMER` = 5u, `PD_GOOD_CRC_TX_TIMER` = 6u,
 `PD_HARD_RESET_TX_TIMER` = 7u,
 `PD_VCONN_SWAP_INITIATOR_TIMER` = 8u, `PD_GENERIC_TIMER` = 9u, `PD_PPS_TIMER` = 10u,
 `PD_SINK_TX_TIMER` = 11u,
 `PD_TIMERS_END_ID` = 14u, `TYPEC_TIMERS_START_ID` = 15u, `TYPEC_CC1_DEBOUNCE_TIMER` =
 15u, `TYPEC_CC2_DEBOUNCE_TIMER` = 16u,
 `TYPEC_RD_DEBOUNCE_TIMER` = 17u, `TYPEC_VBUS_DISCHARGE_TIMER` = 18u, `TYPEC_ACTIVITY_TIMER`
 = 19u, `TYPEC_GENERIC_TIMER1` = 20u,
 `TYPEC_GENERIC_TIMER2` = 21u, `TYPEC_TIMERS_END_ID` = 24u, `PD_OCP_DEBOUNCE_TIMER` =
 25u, `HPD_RX_ACTIVITY_TIMER_ID` = 26u,
 `PD_VCONN_OCP_DEBOUNCE_TIMER` = 27u, `VBUS_DISCHARGE_SCHEDULE_TIMER` = 29u,
 `APP_TIMERS_START_ID` = 30u }

- enum `app_fault_mask_t` {
`CFG_TABLE_OVP_EN_MASK` = 0x01u, `CFG_TABLE_OCP_EN_MASK` = 0x02u, `CFG_TABLE_UVP_EN_MASK` = 0x04u, `CFG_TABLE_SCP_EN_MASK` = 0x08u,
`CFG_TABLE_VCONN_OCP_EN_MASK` = 0x10u, `CFG_TABLE OTP_EN_MASK` = 0x20u }
- enum `app_swap_resp_t` { `APP_RESP_ACCEPT` = 0u, `APP_RESP_REJECT`, `APP_RESP_WAIT`, `APP_RESP_NOT_SUPPORTED` }
- enum `pd_rev_t` { `PD_REV1` = 0, `PD_REV2`, `PD_REV3`, `PD_REV_RSVD` }
- enum `pd_msg_class_t` {
`PD_CTRL_MSG` = 0, `PD_DATA_MSG`, `PD_EXTD_MSG`, `PD_CABLE_RESET`,
`PD_MSG_RSVD` }
- enum `rdo_type_t` { `FIXED_VAR_RDO` = 0, `BAT_RDO` }
- enum `ctrl_msg_t` {
`CTRL_MSG_RSRVD` = 0, `CTRL_MSG_GOOD_CRC`, `CTRL_MSG_GO_TO_MIN`, `CTRL_MSG_ACCEPT`,
`CTRL_MSG_REJECT`, `CTRL_MSG_PING`, `CTRL_MSG_PS_RDY`, `CTRL_MSG_GET_SOURCE_CAP`,
`CTRL_MSG_GET_SINK_CAP`, `CTRL_MSG_DR_SWAP`, `CTRL_MSG_PR_SWAP`, `CTRL_MSG_VCONN_SWAP`,
`CTRL_MSG_WAIT`, `CTRL_MSG_SOFT_RESET`, `CTRL_MSG_NOT_SUPPORTED` = 16, `CTRL_MSG_GET_SRC_CAP_EXTD`,
`CTRL_MSG_GET_STATUS`, `CTRL_MSG_FR_SWAP`, `CTRL_MSG_GET_PPS_STATUS`, `CTRL_MSG_GET_COUNTRY_CO`
}
- enum `data_msg_t` {
`DATA_MSG_SRC_CAP` = 1, `DATA_MSG_REQUEST`, `DATA_MSG_BIST`, `DATA_MSG_SNK_CAP`,
`DATA_MSG_BAT_STATUS`, `DATA_MSG_ALERT`, `DATA_MSG_GET_COUNTRY_INFO`, `DATA_MSG_VDM`
= 15 }
- enum `extd_msg_t` {
`EXTD_MSG_SRC_CAP_EXTD` = 1, `EXTD_MSG_STATUS`, `EXTD_MSG_GET_BAT_CAP`, `EXTD_MSG_GET_BAT_STATUS`,
`EXTD_MSG_BAT_CAP`, `EXTD_MSG_GET_MANU_INFO`, `EXTD_MSG_MANU_INFO`, `EXTD_MSG_SECURITY_REQ`,
`EXTD_MSG_SECURITY_RESP`, `EXTD_MSG_FW_UPDATE_REQ`, `EXTD_MSG_FW_UPDATE_RESP`,
`EXTD_MSG_PPS_STATUS`,
`EXTD_MSG_COUNTRY_INFO`, `EXTD_MSG_COUNTRY_CODES` }
- enum `pdo_t` { `PDO_FIXED_SUPPLY` = 0, `PDO_BATTERY`, `PDO_VARIABLE_SUPPLY`, `PDO_AUGMENTED` }
- enum `apdo_t` { `APDO_PPS` = 0, `APDO_RSVD1`, `APDO_RSVD2`, `APDO_RSVD3` }
- enum `peak_cur_cap_t` { `IMAX_EQ_IOC` = 0, `IMAX_EQ_130_IOC`, `IMAX_EQ_150_IOC`, `IMAX_EQ_200_IOC` }
- enum `bist_mode_t` {
`BIST_RX_MODE` = 0, `BIST_TX_MODE`, `BIST_RETURN_COUNTERS_MODE`, `BIST_CARRIER_MODE_0`,
`BIST_CARRIER_MODE_1`, `BIST_CARRIER_MODE_2`, `BIST_CARRIER_MODE_3`, `BIST_EYE_PATTERN_MODE`,
`BIST_TEST_DATA_MODE` }
- enum `sop_t` {
`SOP` = 0, `SOP_PRIME`, `SOP_DPRIME`, `SOP_P_DEBUG`,
`SOP_DP_DEBUG`, `HARD_RESET`, `CABLE_RESET`, `SOP_INVALID` }
- enum `port_role_t` { `PRT_ROLE_SINK` = 0, `PRT_ROLE_SOURCE`, `PRT_DUAL` }
- enum `port_type_t` { `PRT_TYPE_UFP` = 0, `PRT_TYPE_DFP`, `PRT_TYPE_DRP` }
- enum `fr_swap_supp_t` { `FR_SWAP_NOT_SUPPORTED` = 0, `FR_SWAP_DEF_USB`, `FR_SWAP_1_5A`,
`FR_SWAP_3A` }
- enum `app_req_status_t` {
`REQ_SEND_HARD_RESET` = 1, `REQ_ACCEPT` = 3, `REQ_REJECT` = 4, `REQ_WAIT` = 12,
`REQ_NOT_SUPPORTED` = 16 }
- enum `resp_status_t` {
`SEQ_ABORTED` = 0, `CMD_FAILED`, `RES_TIMEOUT`, `CMD_SENT`,
`RES_RCV` }
- enum `dpm_pd_cmd_t` {
`DPM_CMD_SRC_CAP_CHNG` = 0, `DPM_CMD_SNK_CAP_CHNG`, `DPM_CMD_SEND_GO_TO_MIN`,
`DPM_CMD_GET_SNK_CAP`,
`DPM_CMD_GET_SRC_CAP`, `DPM_CMD_SEND_HARD_RESET`, `DPM_CMD_SEND_SOFT_RESET`,
`DPM_CMD_SEND_CABLE_RESET`,
`DPM_CMD_SEND_SOFT_RESET_EMCA`, `DPM_CMD_SEND_DR_SWAP`, `DPM_CMD_SEND_PR_SWAP`,
`DPM_CMD_SEND_VCONN_SWAP`,

```

DPM_CMD_SEND_VDM, DPM_CMD_SEND_EXTENDED, DPM_CMD_GET_SRC_CAP_EXTENDED,
DPM_CMD_GET_STATUS,
DPM_CMD_SEND_BATT_STATUS, DPM_CMD_SEND_ALERT, DPM_CMD_SEND_NOT_SUPPORTED,
DPM_CMD_SEND_INVALID = 0xFFu }

• enum pd_devtype_t {
    DEV_SNK = 1, DEV_SRC, DEV_DBG_ACC, DEV_AUD_ACC,
    DEV_PWRD_ACC, DEV_UNSUPORTED_ACC }

• enum vdm_type_t { VDM_TYPE_UNSTRUCTURED = 0, VDM_TYPE_STRUCTURED }

• enum std_vdm_cmd_t {
    VDM_CMD_DSC_IDENTITY = 1, VDM_CMD_DSC_SVIDS, VDM_CMD_DSC_MODES, VDM_CMD_ENTER_MODE,
    VDM_CMD_EXIT_MODE, VDM_CMD_ATTENTION, VDM_CMD_DP_STATUS_UPDT = 16, VDM_CMD_DP_CONFIGURE
    = 17 }

• enum std_vdm_cmd_type_t { CMD_TYPE_INITIATOR = 0, CMD_TYPE_RESP_ACK, CMD_TYPE_RESP_NAK,
    CMD_TYPE_RESP_BUSY }

• enum std_vdm_prod_t {
    PROD_TYPE_UNDEF = 0, PROD_TYPE_HUB, PROD_TYPE_PERI, PROD_TYPE_PAS_CBL,
    PROD_TYPE_ACT_CBL, PROD_TYPE_AMA }

• enum std_vdm_ver_t { STD_VDM_VER1 = 0, STD_VDM_VER2, STD_VDM_VER3, STD_VDM_VER4 }

• enum cbl_vbus_cur_t { CBL_VBUS_CUR_0A = 0, CBL_VBUS_CUR_3A, CBL_VBUS_CUR_5A }

• enum cbl_term_t { CBL_TERM_BOTH_PAS_VCONN_NOT_REQ = 0, CBL_TERM_BOTH_PAS_VCONN_REQ,
    CBL_TERM_ONE_ACT_ONE_PAS_VCONN_REQ, CBL_TERM_BOTH_ACT_VCONN_REQ }

• enum pe_cbl_state_t { CBL_FSM_DISABLED = 0, CBL_FSM_ENTRY, CBL_FSM_SEND_SOFT_RESET,
    CBL_FSM_SEND_DSC_ID }

• enum rp_cc_status_t { RP_RA = 0, RP_RD, RP_OPEN }

• enum rd_cc_status_t {
    RD_RA = 0, RD_USB, RD_1_5A, RD_3A,
    RD_ERR }

• enum rp_term_t { RP_TERM_RP_CUR_DEF = 0, RP_TERM_RP_CUR_1_5A, RP_TERM_RP_CUR_3A }

• enum try_src_snk_t { TRY_SRC_TRY_SNK_DISABLED = 0, TRY_SRC_ENABLED, TRY_SNK_ENABLED
    }

• enum dpm_typec_cmd_t {
    DPM_CMD_SET_RP_DFLT = 0, DPM_CMD_SET_RP_1_5A, DPM_CMD_SET_RP_3A, DPM_CMD_PORT_DISABLE,
    DPM_CMD_TYPEC_ERR_RECOVERY, DPM_CMD_TYPEC_INVALID }

• enum dpm_typec_cmd_resp_t { DPM_RESP_FAIL = 0, DPM_RESP_SUCCESS }

• enum typec_fsm_state_t {
    TYPEC_FSM_DISABLED = 0, TYPEC_FSM_ERR_RECov, TYPEC_FSM_ATTACH_WAIT, TYPEC_FSM_TRY_SRC,
    TYPEC_FSM_TRY_WAIT_SNK, TYPEC_FSM_TRY_SNK, TYPEC_FSM_TRY_WAIT_SRC, TYPEC_FSM_UNATTACHED_SNK,
    TYPEC_FSM_UNATTACHED_SNK, TYPEC_FSM_AUD_ACC, TYPEC_FSM_DBG_ACC, TYPEC_FSM_ATTACHED_SRC,
    TYPEC_FSM_ATTACHED_SNK, TYPEC_FSM_MAX_STATES }

• enum pe_fsm_state_t {
    PE_FSM_OFF = 0, PE_FSM_HR_SEND, PE_FSM_HR_SRC_TRANS_DFLT, PE_FSM_HR_SRC_RECOVER,
    PE_FSM_HR_SRC_VBUS_ON, PE_FSM_HR_SNK_TRANS_DFLT, PE_FSM_HR_SNK_WAIT_VBUS_OFF,
    PE_FSM_HR_SNK_WAIT_VBUS_ON,
    PE_FSM_BIST_TEST_DATA, PE_FSM_BIST_CM2, PE_FSM_SNK_STARTUP, PE_FSM_SNK_WAIT_FOR_CAP,
    PE_FSM_SNK_EVAL_CAP, PE_FSM_SNK_SEL_CAP, PE_FSM_SRC_STARTUP, PE_FSM_SRC_WAIT_NEW_CAP,
    PE_FSM_SRC_SEND_CBL_SR, PE_FSM_SRC_SEND_CBL_DSCID, PE_FSM_SRC_SEND_CAP,
    PE_FSM_SRC_DISCOVERY,
    PE_FSM_SRC_NEG_CAP, PE_FSM_SRC_TRANS_SUPPLY, PE_FSM_SRC_SEND_PS_RDY, PE_FSM_SNK_TRANS,
    PE_FSM_SR_SEND, PE_FSM_SR_RECV, PE_FSM_VRS_VCONN_ON, PE_FSM_VRS_VCONN_OFF,
    PE_FSM_SWAP_EVAL, PE_FSM_SWAP_SEND, PE_FSM_DRS_CHANGE_ROLE, PE_FSM_PRS_SRC_SNK_TRANS,
    PE_FSM_PRS_SRC_SNK_VBUS_OFF, PE_FSM_PRS_SRC_SNK_WAIT_PS_RDY, PE_FSM_PRS_SNK_SRC_WAIT_PS_RDY,
    PE_FSM_PRS_SNK_SRC_VBUS_ON,
    PE_FSM_FRS_CHECK_RP, PE_FSM_FRS_SRC_SNK_CC_SIGNAL, PE_FSM_READY, PE_FSM_SEND_MSG,
    PE_FSM_MAX_STATES }

• enum pd_contract_status_t {
    PD_CONTRACT_NEGOTIATION_SUCCESSFUL = 0x01, PD_CONTRACT_CAP_MISMATCH_DETECTED
    = 0x03, PD_CONTRACT_REJECT_CONTRACT_VALID = 0x00, PD_CONTRACT_REJECT_CONTRACT_NOT_VALID

```

```

= 0x04,
PD_CONTRACT_REJECT_NO_CONTRACT = 0x08, PD_CONTRACT_REJECT_EXPLICIT_CONTRACT =
0x0C, PD_CONTRACT_REJECT_NO_EXPLICIT_CONTRACT = 0x10, PD_CONTRACT_PS_READY_NOT_RECEIVED
= 0x14,
PD_CONTRACT_PS_READY_NOT_SENT = 0x18 }
• enum app_evt_t {
APP_EVT_UNEXPECTED_VOLTAGE_ON_VBUS, APP_EVT_TYPE_C_ERROR_RECOVERY, APP_EVT_CONNECT,
APP_EVT_DISCONNECT,
APP_EVT_EMCA_DETECTED, APP_EVT_EMCA_NOT_DETECTED, APP_EVT_ALT_MODE, APP_EVT_APP_HW,
APP_EVT_BB, APP_EVT_RP_CHANGE, APP_EVT_HARD_RESET_RCVD, APP_EVT_HARD_RESET_COMPLETE,
APP_EVT_PKT_RCVD, APP_EVT_PR_SWAP_COMPLETE, APP_EVT_DR_SWAP_COMPLETE,
APP_EVT_VCONN_SWAP_COMPLETE,
APP_EVT_SENDER_RESPONSE_TIMEOUT, APP_EVT_VENDOR_RESPONSE_TIMEOUT, APP_EVT_HARD_RESET_SE,
APP_EVT_SOFT_RESET_SENT,
APP_EVT_CBL_RESET_SENT, APP_EVT_PE_DISABLED, APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE,
APP_EVT_VBUS_OVP_FAULT,
APP_EVT_VBUS_OCP_FAULT, APP_EVT_VCONN_OCP_FAULT, APP_EVT_VBUS_PORT_DISABLE,
APP_EVT_TYPEC_STARTED,
APP_EVT_FR_SWAP_COMPLETE, APP_EVT_TEMPERATUREFAULT, APP_EVT_HANDLE_EXTENDED_MSG,
APP_EVT_VBUS_UVP_FAULT,
APP_EVT_VBUS_SCP_FAULT, APP_EVT_TYPEC_ATTACH_WAIT, APP_EVT_TYPEC_ATTACH_WAIT_TO_UNATTACHE,
APP_EVT_TYPEC_ATTACH,
APP_EVT_CC_OVP, APP_EVT_SBU_OVP, APP_EVT_ALERT_RECEIVED, APP_EVT_SRC_CAP_TRIED_WITH_NO_RESP,
APP_EVT_PD_SINK_DEVICE_CONNECTED }
• enum pd_ams_type { PD_AMS_NONE = 0, PD_AMS_NON_INTR, PD_AMS_INTR }
• enum vdm_ams_t { VDM_AMS_RESP_READY = 0, VDM_AMS_RESP_NOT_REQ, VDM_AMS_RESP_FROM_EC,
VDM_AMS_RESP_NOT_SUPP }

```

Functions

- const `pd_config_t * get_pd_config` (void)
- const `pd_port_config_t * get_pd_port_config` (uint8_t port)
- bool `pd_is_msg` (const `pd_packet_t *pkt`, `sop_t sop_type`, `pd_msg_class_t msg_class`, `uint32_t msg_mask`, `uint16_t length`)
- `ovp_settings_t * pd_get_ptr_ovp_tbl` (uint8_t port)
- `ocp_settings_t * pd_get_ptr_ocp_tbl` (uint8_t port)
- `uvp_settings_t * pd_get_ptr_uvp_tbl` (uint8_t port)
- `scp_settings_t * pd_get_ptr_scp_tbl` (uint8_t port)
- `vconn_ocp_settings_t * pd_get_ptr_vconn_ocp_tbl` (uint8_t port)
- `otp_settings_t * pd_get_ptr_otp_tbl` (uint8_t port)
- `pwr_params_t * pd_get_ptr_pwr_tbl` (uint8_t port)
- `chg_cfg_params_t * pd_get_ptr_chg_cfg_tbl` (uint8_t port)
- `bat_chg_params_t * pd_get_ptr_bat_chg_tbl` (uint8_t port)
- `pwr_params_t * pd_get_ptr_type_a_pwr_tbl` (uint8_t port)
- `typeA_chg_cfg_params_t * pd_get_ptr_type_a_chg_cfg_tbl` (uint8_t port)
- `bb_settings_t * pd_get_ptr_bb_tbl` (uint8_t port)

Variables

- const uint16_t **CUR_TABLE** [5]

6.20.1 Detailed Description

Common definitions and structures used in the CCG USB-PD stack.

Note: Changing values in this header file are not likely to have an impact on the final application behavior; as the pre-defined values would have been compiled into the stack libraries.

6.20.2 Macro Definition Documentation

6.20.2.1 BC_SINK_1_2_MODE_ENABLE_MASK

```
#define BC_SINK_1_2_MODE_ENABLE_MASK (0x01)
```

BC 1.2 sink mode enable mask for config table parameter.

6.20.2.2 BC_SINK_APPLE_MODE_ENABLE_MASK

```
#define BC_SINK_APPLE_MODE_ENABLE_MASK (0x02)
```

Apple sink mode enable mask for config table parameter.

6.20.2.3 BC_SRC_1_2_MODE_ENABLE_MASK

```
#define BC_SRC_1_2_MODE_ENABLE_MASK (0x01)
```

BC 1.2 Source mode enable mask for config table parameter.

6.20.2.4 BC_SRC_AFC_MODE_ENABLE_MASK

```
#define BC_SRC_AFC_MODE_ENABLE_MASK (0x08)
```

AFC source mode enable mask for config table parameter.

6.20.2.5 BC_SRC_APPLE_MODE_ENABLE_MASK

```
#define BC_SRC_APPLE_MODE_ENABLE_MASK (0x02)
```

Apple source mode enable mask for config table parameter.

6.20.2.6 BC_SRC_QC_4_0_MODE_ENABLE_MASK

```
#define BC_SRC_QC_4_0_MODE_ENABLE_MASK (0x10)
```

QC 4.0 mode enable mask for config table parameter.

6.20.2.7 BC_SRC_QC_MODE_ENABLE_MASK

```
#define BC_SRC_QC_MODE_ENABLE_MASK (0x04)
```

QC source mode enable mask for config table parameter.

6.20.2.8 BC_SRC_QC_VER_2_CLASS_A_VAL

```
#define BC_SRC_QC_VER_2_CLASS_A_VAL (0u)
```

QC source Version and class mask for config table parameter.

6.20.2.9 BC_SRC_QC_VER_2_CLASS_B_VAL

```
#define BC_SRC_QC_VER_2_CLASS_B_VAL (1u)
```

QC source Version and class mask for config table parameter.

6.20.2.10 BC_SRC_QC_VER_3_CLASS_A_VAL

```
#define BC_SRC_QC_VER_3_CLASS_A_VAL (2u)
```

QC source Version and class mask for config table parameter.

6.20.2.11 BC_SRC_QC_VER_3_CLASS_B_VAL

```
#define BC_SRC_QC_VER_3_CLASS_B_VAL (3u)
```

QC source Version and class mask for config table parameter.

6.20.2.12 BDO_HDR_IDX

```
#define BDO_HDR_IDX (0u)
```

Index of BIST header data object in a received message.

6.20.2.13 CC_CHANNEL_1

```
#define CC_CHANNEL_1 (0u)
```

Reference to CC_1 pin in the Type-C connector.

6.20.2.14 CC_CHANNEL_2

```
#define CC_CHANNEL_2 (1u)
```

Reference to CC_2 pin in the Type-C connector.

6.20.2.15 CCG_CC_STAT_DRP_TOGGLE

```
#define CCG_CC_STAT_DRP_TOGGLE (1u)
```

CC line status: DRP toggle in progress.

6.20.2.16 CCG_CC_STAT_RD_PRESENT

```
#define CCG_CC_STAT_RD_PRESENT (2u)
```

CC line status: Rd is applied.

6.20.2.17 CCG_CC_STAT_RP_PRESENT

```
#define CCG_CC_STAT_RP_PRESENT (4u)
```

CC line status: Rp is applied.

6.20.2.18 CCG_CC_STAT_VCONN_ACTIVE

```
#define CCG_CC_STAT_VCONN_ACTIVE (8u)
```

CC line status: VConn is applied.

6.20.2.19 CCG_CC_STAT_ZOPEN

```
#define CCG_CC_STAT_ZOPEN (0u)
```

CC line status: ZOpen.

6.20.2.20 CCG_DPM_ERROR_NO_VCONN

```
#define CCG_DPM_ERROR_NO_VCONN (1u)
```

DPM command failed due to absence of VConn.

6.20.2.21 CCG_DPM_ERROR_NONE

```
#define CCG_DPM_ERROR_NONE (0u)
```

No additional DPM error information available.

6.20.2.22 CCG_FRS_RX_ENABLE_MASK

```
#define CCG_FRS_RX_ENABLE_MASK (0x01u)
```

FRS receive enable flag in config table setting.

6.20.2.23 CCG_FRS_TX_ENABLE_MASK

```
#define CCG_FRS_TX_ENABLE_MASK (0x02u)
```

FRS transmit enable flag in config table setting.

6.20.2.24 CCG_PD_EXT_PPS_STATUS_SIZE

```
#define CCG_PD_EXT_PPS_STATUS_SIZE (4u)
```

Size of PPS status extended message in bytes.

6.20.2.25 CCG_PD_EXT_SRCCAP_INP_INDEX

```
#define CCG_PD_EXT_SRCCAP_INP_INDEX (21u)
```

Index of Source Inputs field in extended source caps.

6.20.2.26 CCG_PD_EXT_SRCCAP_INP_UNCONSTRAINED

```
#define CCG_PD_EXT_SRCCAP_INP_UNCONSTRAINED (0x02u)
```

Mask for unconstrained source input in extended source caps.

6.20.2.27 CCG_PD_EXT_SRCCAP_PDP_INDEX

```
#define CCG_PD_EXT_SRCCAP_PDP_INDEX (23u)
```

Index of PDP field in extended source caps.

6.20.2.28 CCG_PD_EXT_SRCCAP_SIZE

```
#define CCG_PD_EXT_SRCCAP_SIZE (24u)
```

Size of extended source capabilities message in bytes.

6.20.2.29 CCG_PD_EXT_STATUS_SIZE

```
#define CCG_PD_EXT_STATUS_SIZE (5u)
```

Size of status extended message in bytes.

6.20.2.30 CCG_PD_FIX_SRC_PDO_MASK_REV2

```
#define CCG_PD_FIX_SRC_PDO_MASK_REV2 (0xFE3FFFFFFu)
```

Mask to be applied on Fixed Supply Source PDO for PD Rev 2.0

6.20.2.31 CCG_PD_FIX_SRC_PDO_MASK_REV3

```
#define CCG_PD_FIX_SRC_PDO_MASK_REV3 (0xFF3FFFFFFu)
```

Mask to be applied on Fixed Supply Source PDO for PD Rev 3.0

6.20.2.32 CCG_PD_FLAG_CONTRACT_NEG_ACTIVE

```
#define CCG_PD_FLAG_CONTRACT_NEG_ACTIVE (1u)
```

Status field indicating that contract negotiation is in progress.

6.20.2.33 CCG_PD_FLAG_EXPLICIT_CONTRACT

```
#define CCG_PD_FLAG_EXPLICIT_CONTRACT (2u)
```

Status field indicating that explicit contract is present.

6.20.2.34 CCG_PD_FLAG_POWER_SINK

```
#define CCG_PD_FLAG_POWER_SINK (8u)
```

Status field indicating that the port is currently a sink.

6.20.2.35 CCG_PD_FLAG_SRC_READY

```
#define CCG_PD_FLAG_SRC_READY (4u)
```

Status field indicating that source is ready.

6.20.2.36 CY_VID

```
#define CY_VID (0x04B4u)
```

Cypress VID defined by Cypress for field upgrades.

6.20.2.37 DP_SVID

```
#define DP_SVID (0xFF01u)
```

Displayport SVID defined by VESA specification.

6.20.2.38 DRP_TOGGLE_PERIOD

```
#define DRP_TOGGLE_PERIOD (75u)
```

Minimum DRP toggling period, in ms. See Table 4-16 of the Type-C spec Rev1.

6.20.2.39 GET_DR_SWAP_RESP

```
#define GET_DR_SWAP_RESP (
    resp ) (((resp) & 0x3u)
```

Macro to extract the default DR_SWAP command response from the swap_response field in the configuration table.

6.20.2.40 GET_PR_SWAP_RESP

```
#define GET_PR_SWAP_RESP (
    resp ) (((((resp) & 0xCu) >> 2u)
```

Macro to extract the default PR_SWAP command response from the swap_response field in the configuration table.

6.20.2.41 GET_VCONN_SWAP_RESP

```
#define GET_VCONN_SWAP_RESP (
    resp ) (((((resp) & 0x30u) >> 4u)
```

Macro to extract the default VCONN_SWAP command response from the swap_response field in the configuration table.

6.20.2.42 GIVE_BACK_MASK

```
#define GIVE_BACK_MASK (0x8000u)
```

Mask for the give-back supported bit in the snk_pdo_max_min_current_pwr field of the configuration table.

6.20.2.43 HPD_RX_ACTIVITY_TIMER_PERIOD_MAX

```
#define HPD_RX_ACTIVITY_TIMER_PERIOD_MAX (105u)
```

Maximum HPD receiver timer period in ms.

6.20.2.44 HPD_RX_ACTIVITY_TIMER_PERIOD_MIN

```
#define HPD_RX_ACTIVITY_TIMER_PERIOD_MIN (5u)
```

Minimum HPD receiver timer period in ms.

6.20.2.45 I_1A

```
#define I_1A (100)
```

VBus current usage = 1.0 A.

6.20.2.46 I_1P5A

```
#define I_1P5A (150)
```

VBus current usage = 1.5 A.

6.20.2.47 I_2A

```
#define I_2A (200)
```

VBus current usage = 2.0 A.

6.20.2.48 I_3A

```
#define I_3A (300)
```

VBus current usage = 3.0 A.

6.20.2.49 I_5A

```
#define I_5A (500)
```

VBus current usage = 5.0 A.

6.20.2.50 ID_HEADER_IDX

```
#define ID_HEADER_IDX (1u)
```

Index of ID_HEADER data object in a received VDM.

6.20.2.51 ISAFE_0A

```
#define ISAFE_0A (0u)
```

VBus current usage = 0 A.

6.20.2.52 ISAFE_DEF

```
#define ISAFE_DEF (50u)
```

VBus current usage = 0.5 A.

6.20.2.53 MAX_CBL_DSC_ID_COUNT

```
#define MAX_CBL_DSC_ID_COUNT (20u)
```

Maximum number of cable discovery DISCOVER_IDENTITY messages that should be sent out.

6.20.2.54 MAX_EXTD_MSG_LEGACY_LEN

```
#define MAX_EXTD_MSG_LEGACY_LEN (26u)
```

Maximum legacy Extended message size in bytes.

6.20.2.55 MAX_EXTD_PKT_SIZE

```
#define MAX_EXTD_PKT_SIZE (260u)
```

Maximum extended message size in bytes.

6.20.2.56 MAX_EXTD_PKT_WORDS

```
#define MAX_EXTD_PKT_WORDS (65u)
```

Maximum extended message 32-bit words. Each word is 32 bit.

6.20.2.57 MAX_HARD_RESET_COUNT

```
#define MAX_HARD_RESET_COUNT (3u)
```

Maximum hard reset retry count.

6.20.2.58 MAX_MESSAGE_ID

```
#define MAX_MESSAGE_ID (7u)
```

Maximum message id value in PD Header.

6.20.2.59 MAX_NO_OF_DO

```
#define MAX_NO_OF_DO (7u)
```

Maximum number of DOs in a packet. Limited by PD message definition.

6.20.2.60 MAX_NO_OF_PDO

```
#define MAX_NO_OF_PDO (MAX_NO_OF_DO)
```

Maximum number of PDOs in a packet. Limited by PD message definition.

6.20.2.61 MAX_NO_OF_VDO

```
#define MAX_NO_OF_VDO (MAX_NO_OF_DO)
```

Maximum number of VDOs in a packet. Limited by PD message definition.

6.20.2.62 MAX_SOP_TYPES

```
#define MAX_SOP_TYPES (3)
```

Max SOP types excluding hard reset, cable reset, SOP_PDEBUG and SOP_DPDEBUG.

6.20.2.63 MAX_SRC_CAP_COUNT

```
#define MAX_SRC_CAP_COUNT (50u)
```

Maximum retries of Source Capability messages.

6.20.2.64 PD_BIST_CONT_MODE_TIMER_PERIOD

```
#define PD_BIST_CONT_MODE_TIMER_PERIOD (55u)
```

BIST continuous mode period in ms. See Section 6.5.8.4 of USBPD Spec Rev2 v1.2

6.20.2.65 PD_CBL_DELAY_TIMER_PERIOD

```
#define PD_CBL_DELAY_TIMER_PERIOD (2u)
```

Cable delay timer period in ms. See Section 6.5.14 of USBPD Spec Rev2 v1.2

6.20.2.66 PD_CBL_DSC_ID_START_TIMER_PERIOD

```
#define PD_CBL_DSC_ID_START_TIMER_PERIOD (43u)
```

Cable discovery start period in ms. See Section 6.5.15 of USBPD Spec Rev2 v1.2

6.20.2.67 PD_CBL_DSC_ID_TIMER_PERIOD

```
#define PD_CBL_DSC_ID_TIMER_PERIOD (49u)
```

Cable discovery timer period in ms. See Section 6.5.15 of USBPD Spec Rev2 v1.2

6.20.2.68 PD_CBL_READY_TIMER_PERIOD

```
#define PD_CBL_READY_TIMER_PERIOD (50u)
```

This timer is the delay between PD startup and sending cable Discover ID request to ensure cable is ready to respond.

6.20.2.69 PD_COLLISION_SRC_COOL_OFF_TIMER_PERIOD

```
#define PD_COLLISION_SRC_COOL_OFF_TIMER_PERIOD (5u)
```

Time for which PD 3.0 source will keep Rp as SinkTxNG after returning to Ready state.

6.20.2.70 PD_CUR_PER_UNIT

```
#define PD_CUR_PER_UNIT (10u)
```

Current unit used in PDOs.

6.20.2.71 PD_DPM_RESP_REC_RESP_PERIOD

```
#define PD_DPM_RESP_REC_RESP_PERIOD (20u)
```

VDM receiver response timer period in ms. See Section 6.5.12.1 of USBPD Spec Rev2 v1.2. This timer is slightly relaxed from the spec value.

6.20.2.72 PD_EXTERNALLY_POWERED_BIT_POS

```
#define PD_EXTERNALLY_POWERED_BIT_POS (7u)
```

Externally powered bit position in Source PDO mask.

6.20.2.73 PD_HARD_RESET_TX_TIMER_PERIOD

```
#define PD_HARD_RESET_TX_TIMER_PERIOD (20u)
```

Hard reset transmit timer period in ms. See Section 6.3.13 of USBPD Spec Rev2 v1.2

6.20.2.74 PD_MAX_SRC_CAP_TRY

```
#define PD_MAX_SRC_CAP_TRY (6u)
```

MAX SRC CAP retry count used to determine if the device connected is PD Capable or not.

6.20.2.75 PD_NO_RESPONSE_TIMER_PERIOD

```
#define PD_NO_RESPONSE_TIMER_PERIOD (5000u)
```

PD No response timer period in ms. See Section 6.5.7 of USBPD Spec Rev2 v1.2

6.20.2.76 PD_PHY_BUSY_TIMER_PERIOD

```
#define PD_PHY_BUSY_TIMER_PERIOD (15u)
```

Period of timer used internally by stack to prevent PHY lockup in TX state.

6.20.2.77 PD_PPS_SRC_TIMER_PERIOD

```
#define PD_PPS_SRC_TIMER_PERIOD (14000u)
```

PPS timer period in ms.

6.20.2.78 PD_PS_HARD_RESET_TIMER_PERIOD

```
#define PD_PS_HARD_RESET_TIMER_PERIOD (27u)
```

Hard reset timer period in ms. See Section 6.5.11.2 of USBPD Spec Rev2 v1.2

6.20.2.79 PD_PS_SNK_TRANSITION_TIMER_PERIOD

```
#define PD_PS_SNK_TRANSITION_TIMER_PERIOD (500u)
```

Snk. transition period in ms. See Section 6.5.6.1 of USBPD Spec Rev2 v1.2

6.20.2.80 PD_PS_SRC_OFF_TIMER_PERIOD

```
#define PD_PS_SRC_OFF_TIMER_PERIOD (900u)
```

Src. off timer period in ms. See Section 6.5.6.2 of USBPD Spec Rev2 v1.2

6.20.2.81 PD_PS_SRC_ON_TIMER_PERIOD

```
#define PD_PS_SRC_ON_TIMER_PERIOD (450u)
```

Src. on timer period in ms. See Section 6.5.6.3 of USBPD Spec Rev2 v1.2

6.20.2.82 PD_PS_SRC_TRANS_TIMER_PERIOD

```
#define PD_PS_SRC_TRANS_TIMER_PERIOD (400u)
```

Src.Trans timer period in ms. See Section 6.5.6.1 of USBPD Spec Rev2 v1.2

6.20.2.83 PD_RECEIVER_RESPONSE_TIMER_PERIOD

```
#define PD_RECEIVER_RESPONSE_TIMER_PERIOD (15u)
```

Receiver response timeout period in ms. See Section 6.5.2 of USBPD Spec Rev2 v1.2

6.20.2.84 PD_SENDER_RESPONSE_TIMER_PERIOD

```
#define PD_SENDER_RESPONSE_TIMER_PERIOD (27u)
```

Sender response timeout period in ms. See Section 6.5.2 of USBPD Spec Rev2 v1.2

6.20.2.85 PD_SINK_TX_TIMER_PERIOD

```
#define PD_SINK_TX_TIMER_PERIOD (18u)
```

Delay between AMS initiation attempts by PD 3.0 sink while Rp = SinkTxNG.

6.20.2.86 PD_SINK_VBUS_TURN_OFF_TIMER_PERIOD

```
#define PD_SINK_VBUS_TURN_OFF_TIMER_PERIOD (750u)
```

Time in ms allowed for VBus turn off during hard reset.

6.20.2.87 PD_SINK_VBUS_TURN_ON_TIMER_PERIOD

```
#define PD_SINK_VBUS_TURN_ON_TIMER_PERIOD (1300u)
```

Time in ms allowed for VBus to turn on during hard reset.

6.20.2.88 PD_SINK_WAIT_CAP_TIMER_PERIOD

```
#define PD_SINK_WAIT_CAP_TIMER_PERIOD (400u)
```

Snk wait cap period in ms. See Section 6.5.4.2 of USBPD Spec Rev2 v1.2

6.20.2.89 PD_SOURCE_TRANSITION_TIMER_PERIOD

```
#define PD_SOURCE_TRANSITION_TIMER_PERIOD (28u)
```

Source voltage transition timer period in ms. See Table 7-22 of USBPD Spec Rev2 v1.2

6.20.2.90 PD_SRC_CAP_TIMER_PERIOD

```
#define PD_SRC_CAP_TIMER_PERIOD (150u)
```

Src cap timer period in ms. See Section 6.5.4.1 of USBPD Spec Rev2 v1.2

6.20.2.91 PD_SRC_RECOVER_TIMER_PERIOD

```
#define PD_SRC_RECOVER_TIMER_PERIOD (800u)
```

Src Recover timer period in ms. See Section 7.6.1 of USBPD Spec Rev2 v1.2

6.20.2.92 PD_SWAP_SRC_START_TIMER_PERIOD

```
#define PD_SWAP_SRC_START_TIMER_PERIOD (55u)
```

Src start (during PR_SWAP) period in ms. See Section 6.5.9.2 of USBPD Spec Rev2 v1.2

6.20.2.93 PD_VBUS_TURN_OFF_TIMER_PERIOD

```
#define PD_VBUS_TURN_OFF_TIMER_PERIOD (625u)
```

VBus OFF timer period in ms. See Table 7-22 of USBPD Spec Rev2 v1.2

6.20.2.94 PD_VBUS_TURN_ON_TIMER_PERIOD

```
#define PD_VBUS_TURN_ON_TIMER_PERIOD (275u)
```

VBus ON timer period in ms. See Table 7-22 of USBPD Spec Rev2 v1.2

6.20.2.95 PD_VCONN_OFF_TIMER_PERIOD

```
#define PD_VCONN_OFF_TIMER_PERIOD (25u)
```

Vconn off timer period in ms. See Section 6.5.13 of USBPD Spec Rev2 v1.2

6.20.2.96 PD_VCONN_ON_TIMER_PERIOD

```
#define PD_VCONN_ON_TIMER_PERIOD (100u)
```

VConn on timer period in ms.

6.20.2.97 PD_VCONN_SWAP_INITIATOR_TIMER_PERIOD

```
#define PD_VCONN_SWAP_INITIATOR_TIMER_PERIOD (110u)
```

This timer used by stack to do auto retry VCONN swap before PR swap (if DUT is sink). Minimum gap between VCONN swap request shall be a minimum 100ms, to be safe 110ms is used.

6.20.2.98 PD_VCONN_TURN_ON_TIMER_PERIOD

```
#define PD_VCONN_TURN_ON_TIMER_PERIOD (10u)
```

Period of VConn monitoring checks done internally.

6.20.2.99 PD_VDM_ENTER_MODE_RESPONSE_TIMER_PERIOD

```
#define PD_VDM_ENTER_MODE_RESPONSE_TIMER_PERIOD (45u)
```

Enter mode response timeout period in ms. See Section 6.5.12.2 of USBPD Spec Rev2 v1.2

6.20.2.100 PD_VDM_EXIT_MODE_RESPONSE_TIMER_PERIOD

```
#define PD_VDM_EXIT_MODE_RESPONSE_TIMER_PERIOD (45u)
```

Exit mode response timeout period in ms. See Section 6.5.12.3 of USBPD Spec Rev2 v1.2

6.20.2.101 PD_VDM_RESPONSE_TIMER_PERIOD

```
#define PD_VDM_RESPONSE_TIMER_PERIOD (25u)
```

VDM response timer period in ms. See Section 6.5.12.1 of USBPD Spec Rev2 v1.2

6.20.2.102 PD_VOLT_PER_UNIT

```
#define PD_VOLT_PER_UNIT (50u)
```

Voltage unit used in PDOs.

6.20.2.103 RDO_IDX

```
#define RDO_IDX (0u)
```

Index of Request data object in a received message.

6.20.2.104 SNK_DETACH_VBUS_POLL_COUNT

```
#define SNK_DETACH_VBUS_POLL_COUNT (5u)
```

Number of VBus checks used to detect detach as sink.

6.20.2.105 SNK_MIN_MAX_MASK

```
#define SNK_MIN_MAX_MASK (0x3FFu)
```

Mask to extract the actual min/max current value from the snk_pdo_max_min_current_pwr field of the configuration table.

6.20.2.106 SRC_DRP_MIN_DC

```
#define SRC_DRP_MIN_DC (30)
```

Minimum percentage of DRP period for a source. See Table 4-16 of the Type-C spec Rev1.

6.20.2.107 STD_SVID

```
#define STD_SVID (0xFF00u)
```

Standard SVID defined by USB-PD specification.

6.20.2.108 STD_VDM_VERSION

```
#define STD_VDM_VERSION (0u)
```

Default VDM version used. Corresponds to VDM version 1.0.

6.20.2.109 STD_VDM_VERSION_IDX

```
#define STD_VDM_VERSION_IDX (13u)
```

Position of VDM version field in structured VDM header.

6.20.2.110 STD_VDM_VERSION_REV2

```
#define STD_VDM_VERSION_REV2 (0u)
```

VDM version 1.0. Used under USB-PD Revision 2.0.

6.20.2.111 STD_VDM_VERSION_REV3

```
#define STD_VDM_VERSION_REV3 (1u)
```

VDM version 2.0. Used under USB-PD Revision 3.0.

6.20.2.112 TBT_SVID

```
#define TBT_SVID (0x8087u)
```

Thunderbolt SVID defined by Intel specification.

6.20.2.113 TYPEC_CC_DEBOUNCE_TIMER_PERIOD

```
#define TYPEC_CC_DEBOUNCE_TIMER_PERIOD (140u)
```

CC debounce period in ms from Type-C spec.

6.20.2.114 TYPEC_DRP_TRY_TIMER_PERIOD

```
#define TYPEC_DRP_TRY_TIMER_PERIOD (110u)
```

Type-C Try DRP timer period in ms.

6.20.2.115 TYPEC_ERROR_RECOVERY_TIMER_PERIOD

```
#define TYPEC_ERROR_RECOVERY_TIMER_PERIOD (50u)
```

Type-C error recovery timer period in ms.

6.20.2.116 TYPEC_FSM_GENERIC

```
#define TYPEC_FSM_GENERIC (0x00000001u)
```

Type-C state machine active mode.

6.20.2.117 TYPEC_FSM_NONE

```
#define TYPEC_FSM_NONE (0x00000000u)
```

Type-C state machine inactive mode.

6.20.2.118 TYPEC_PD3_RPCHANGE_DEBOUNCE_PERIOD

```
#define TYPEC_PD3_RPCHANGE_DEBOUNCE_PERIOD (2u)
```

Period in ms used to detect Rp change in a PD 3.0 contract.

6.20.2.119 TYPEC_PD_DEBOUNCE_TIMER_PERIOD

```
#define TYPEC_PD_DEBOUNCE_TIMER_PERIOD (11u)
```

PD debounce period in ms.

6.20.2.120 TYPEC_RD_DEBOUNCE_TIMER_PERIOD

```
#define TYPEC_RD_DEBOUNCE_TIMER_PERIOD (12u)
```

Rd debounce period (detach detection) in ms.

6.20.2.121 TYPEC_SRC_DETACH_DEBOUNCE_PERIOD

```
#define TYPEC_SRC_DETACH_DEBOUNCE_PERIOD (2u)
```

Debounce period used to detect detach when we are source.

6.20.2.122 VDM_HEADER_IDX

```
#define VDM_HEADER_IDX (0u)
```

Index of VDM header data object in a received message.

6.20.2.123 VSAFE_0V

```
#define VSAFE_0V (800u)
```

Vbus voltage = 0.8 V

6.20.2.124 VSAFE_0V_HARD_RESET

```
#define VSAFE_0V_HARD_RESET (3000u)
```

Maximum voltage allowed at the end of a Hard Reset when CCGx is SNK. This is set to 3.0 V.

6.20.2.125 VSAFE_0V_PR_SWAP_SNK_SRC

```
#define VSAFE_0V_PR_SWAP_SNK_SRC (3000u)
```

Vbus voltage = 3.0 V

6.20.2.126 VSAFE_12V

```
#define VSAFE_12V (12000u)
```

Vbus voltage = 12.0 V

6.20.2.127 VSAFE_13V

```
#define VSAFE_13V (13000u)
```

Vbus voltage = 13.0 V

6.20.2.128 VSAFE_15V

```
#define VSAFE_15V (15000u)
```

Vbus voltage = 15.0 V

6.20.2.129 VSAFE_19V

```
#define VSAFE_19V (19000u)
```

Vbus voltage = 19.0 V

6.20.2.130 VSAFE_20V

```
#define VSAFE_20V (20000u)
```

Vbus voltage = 20.0 V

6.20.2.131 VSAFE_3_6V

```
#define VSAFE_3_6V (3600u)
```

Vbus voltage = 3.6 V

6.20.2.132 VSAFE_5V

```
#define VSAFE_5V (5000u)
```

Vbus voltage = 5.0 V

6.20.2.133 VSAFE_9V

```
#define VSAFE_9V (9000u)
```

Vbus voltage = 9.0 V

6.20.3 Typedef Documentation

6.20.3.1 app_resp_cbk_t

```
typedef void(* app_resp_cbk_t) (uint8_t port, app_resp_t *resp)
```

Application response callback. This is the type of callback used by the stack to receive application level responses associated with a PD message such as a SWAP request.

Parameters

<i>port</i>	PD port index.
<i>resp</i>	Pointer to the structure holding response information.

6.20.3.2 dpm_pd_cmd_cbk_t

```
typedef void(* dpm_pd_cmd_cbk_t) (uint8_t port, resp_status_t resp, const pd_packet_t *pkt_ptr)
```

DPM PD command callback. This is the type of callback function used by the Policy Engine to report results of a command to the application layer.

Parameters

<i>port</i>	PD port index.
<i>resp</i>	Response code.
<i>pkt_ptr</i>	Pointer to any PD packet associated with the response.

6.20.3.3 dpm_typec_cmd_cbk_t

```
typedef void(* dpm_typec_cmd_cbk_t) (uint8_t port, dpm_typec_cmd_resp_t resp)
```

Type C command response callback. Type of callback used by the stack to report results of a dpm_typec_command API call to the application layer.

Parameters

<i>port</i>	PD port index.
<i>resp</i>	Response code.

6.20.3.4 pd_cbk_t

```
typedef void(* pd_cbk_t) (uint8_t port, uint32_t event)
```

PD callback prototype. This is a stack internal callback function used by the USB-PD Protocol layer to send events to the Policy Engine. The events notified correspond to Policy Engine events such as HARD RESET or SOFT RESET received.

Parameters

<i>port</i>	PD port index.
<i>Type</i>	of event being notified.

6.20.3.5 pwr_ready_cbk_t

```
typedef void(* pwr_ready_cbk_t) (uint8_t port)
```

Power ready callback. Type of callback used by the stack to receive notification from the power source/sink hardware manager that the requested power transition has been completed.

Parameters

<i>port</i>	PD port index.
-------------	----------------

6.20.3.6 sink_discharge_off_cbk_t

```
typedef void(* sink_discharge_off_cbk_t) (uint8_t port)
```

Sink discharge off callback. Callback type used by the stack to receive notification that the sink discharge circuit has been turned off.

Parameters

<i>port</i>	PD port index.
-------------	----------------

6.20.3.7 vdm_resp_cbk_t

```
typedef void(* vdm_resp_cbk_t) (uint8_t port, vdm\_resp\_t *resp)
```

VDM response callback. This is the type of callback used by the stack to receive application level responses to a VDM received from the port partner or cable marker.

Parameters

<i>port</i>	PD port index.
<i>resp</i>	Pointer to structure holding response information.

6.20.4 Enumeration Type Documentation**6.20.4.1 apdo_t**

```
enum apdo\_t
```

Enum of the Augmented PDO types.

Enumerator

APDO_PPS	Programmable Power Supply PDO.
APDO_RSVD1	Reserved for future use.
APDO_RSVD2	Reserved for future use.
APDO_RSVD3	Reserved for future use.

6.20.4.2 app_evt_t

```
enum app_evt_t
```

Enum of events that are signalled to the application.

Enumerator

APP_EVT_UNEXPECTED_VOLTAGE_ON_VBUS	0x00: Unexpected high voltage seen on VBus.
APP_EVT_TYPE_C_ERROR_RECOVERY	0x01: Type-C error recovery initiated.
APP_EVT_CONNECT	0x02: Type-C connect detected.
APP_EVT_DISCONNECT	0x03: Type-C disconnect(detach) detected.
APP_EVT_EMCA_DETECTED	0x04: Cable (EMCA) discovery successful.
APP_EVT_EMCA_NOT_DETECTED	0x05: Cable (EMCA) discovery timed out.
APP_EVT_ALT_MODE	0x06: Alternate mode related event.
APP_EVT_APP_HW	0x07: MUX control related event.
APP_EVT_BB	0x08: Billboard status change.
APP_EVT_RP_CHANGE	0x09: Rp termination change detected.
APP_EVT_HARD_RESET_RCVD	0x0A: Hard Reset received.
APP_EVT_HARD_RESET_COMPLETE	0x0B: Hard Reset processing completed.
APP_EVT_PKT_RCVD	0x0C: New PD message received.
APP_EVT_PR_SWAP_COMPLETE	0x0D: PR_SWAP process completed.
APP_EVT_DR_SWAP_COMPLETE	0x0E: DR_SWAP process completed.
APP_EVT_VCONN_SWAP_COMPLETE	0x0F: VConn_SWAP process completed.
APP_EVT_SENDER_RESPONSE_TIMEOUT	0x10: Sender response timeout occurred.
APP_EVT_VENDOR_RESPONSE_TIMEOUT	0x11: Vendor message response timeout occurred.
APP_EVT_HARD_RESET_SENT	0x12: Hard Reset sent by CCG.
APP_EVT_SOFT_RESET_SENT	0x13: Soft Reset sent by CCG.
APP_EVT_CBL_RESET_SENT	0x14: Cable Reset sent by CCG.
APP_EVT_PE_DISABLED	0x15: PE.Disabled state entered.
APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE	0x16: Contract negotiation completed.
APP_EVT_VBUS_OVP_FAULT	0x17: VBus Over Voltage fault detected.
APP_EVT_VBUS_OCP_FAULT	0x18: VBus Over Current fault detected.
APP_EVT_VCONN_OCP_FAULT	0x19: VConn Over Current fault detected.
APP_EVT_VBUS_PORT_DISABLE	0x1A: PD port disable completed.
APP_EVT_TYPEC_STARTED	0x1B: PD port enable (start) completed.
APP_EVT_FR_SWAP_COMPLETE	0x1C: FR_SWAP process completed.
APP_EVT_TEMPERATURE_FAULT	0x1D: Over Temperature fault detected.
APP_EVT_HANDLE_EXTENDED_MSG	0x1E: Extended message received and needs to be handled.

Enumerator

APP_EVT_VBUS_UVP_FAULT	0x1F: VBus Under Voltage fault detected.
APP_EVT_VBUS_SCP_FAULT	0x20: VBus Short Circuit fault detected.
APP_EVT_TYPEC_ATTACH_WAIT	0x21: Type-C AttachWait state entered. For internal use only.
APP_EVT_TYPEC_ATTACH_WAIT_TO_UNATTACHED	0x22: Type-C transition from AttachWait to Unattached. For internal use only.
APP_EVT_TYPEC_ATTACH	0x23: Type-C attach event.
APP_EVT_CC_OVP	0x24: Over Voltage on CC/VConn line detected.
APP_EVT_SBU_OVP	0x25: Over Voltage on SBU1/SBU2 line detected.
APP_EVT_ALERT_RECEIVED	0x26: Alert message received. For internal use only.
APP_EVT_SRC_CAP_TRIED_WITH_NO_RESPONSE	0x27: Src Cap tried with no response. For internal use only.
APP_EVT_PD_SINK_DEVICE_CONNECTED	0x28: Sink device connected. For internal use only.

6.20.4.3 app_fault_mask_t

```
enum app_fault_mask_t
```

Fault handling enable/disable masks for use in the configuration table. The respective fault handling will only be enabled if the corresponding bit is set in the protection_enable bit in the configuration table.

Enumerator

CFG_TABLE_OVP_EN_MASK	VBUS Over-Voltage fault handling enable mask.
CFG_TABLE_OCP_EN_MASK	VBUS Over-Current fault handling enable mask.
CFG_TABLE_UVP_EN_MASK	VBUS Under-Voltage fault handling enable mask.
CFG_TABLE_SCP_EN_MASK	VBUS Short-Circuit fault handling enable mask.
CFG_TABLE_VCONN_OCP_EN_MASK	VCONN Over-Current fault handling enable mask.
CFG_TABLE_OTP_EN_MASK	Over-Temperature fault handling enable mask.

6.20.4.4 app_req_status_t

```
enum app_req_status_t
```

Enum of the PD Request results. Enum fields map to the control message field in the PD spec.

Enumerator

REQ_SEND_HARD_RESET	Invalid message. Send Hard Reset.
REQ_ACCEPT	Send Accept message.
REQ_REJECT	Send Reject message.
REQ_WAIT	Send Wait message.
REQ_NOT_SUPPORTED	Send Not_Supported message. Will translate to Reject message under PD 2.0

6.20.4.5 app_swap_resp_t

```
enum app_swap_resp_t
```

Possible responses to various USB-PD swap requests from the application layer. The PD stack hands the requests up to the application for handling and gets directions on handling the request in the form of these response codes.

Enumerator

APP_RESP_ACCEPT	Swap request should be accepted.
APP_RESP_REJECT	Swap request should be rejected.
APP_RESP_WAIT	Swap request handling should be delayed (send wait response).
APP_RESP_NOT_SUPPORTED	Swap request is not supported.

6.20.4.6 bist_mode_t

```
enum bist_mode_t
```

Enum of the BIST modes.

Enumerator

BIST_RX_MODE	BIST receiver mode.
BIST_TX_MODE	BIST transmit mode.
BIST_RETURN_COUNTERS_MODE	Send Returned BIST counters response.
BIST_CARRIER_MODE_0	BIST carrier mode 0.
BIST_CARRIER_MODE_1	BIST carrier mode 1.
BIST_CARRIER_MODE_2	BIST carrier mode 2.
BIST_CARRIER_MODE_3	BIST carrier mode 3.
BIST_EYE_PATTERN_MODE	BIST eye pattern.
BIST_TEST_DATA_MODE	BIST test data mode.

6.20.4.7 cbl_term_t

```
enum cbl_term_t
```

Enum of the cable termination types.

Enumerator

CBL_TERM_BOTH_PAS_VCONN_NOT_REQ	Passive cable, VConn not required.
CBL_TERM_BOTH_PAS_VCONN_REQ	Passive cable, VConn required.
CBL_TERM_ONE_ACT_ONE_PAS_VCONN_REQ	One end active, one end passive, VConn required.
CBL_TERM_BOTH_ACT_VCONN_REQ	Both ends of cable are active, VConn required.

6.20.4.8 cbl_vbus_cur_t

enum `cbl_vbus_cur_t`

Enum of the cable current levels.

Enumerator

CBL_VBUS_CUR_0A	Cable does not conduct VBus power through.
CBL_VBUS_CUR_3A	Cable can support a maximum of 3A.
CBL_VBUS_CUR_5A	Cable can support a maximum of 5A.

6.20.4.9 ctrl_msg_t

enum `ctrl_msg_t`

Enum of the control message types.

Enumerator

CTRL_MSG_RSRVD	0x00: Reserved message code.
CTRL_MSG_GOOD_CRC	0x01: GoodCRC message.
CTRL_MSG_GO_TO_MIN	0x02: GotoMin message.
CTRL_MSG_ACCEPT	0x03: Accept message.
CTRL_MSG_REJECT	0x04: Reject message.
CTRL_MSG_PING	0x05: Ping message.
CTRL_MSG_PS_RDY	0x06: PS_RDY message.
CTRL_MSG_GET_SOURCE_CAP	0x07: Get_Source_Cap message.
CTRL_MSG_GET_SINK_CAP	0x08: Get_Sink_Cap message.
CTRL_MSG_DR_SWAP	0x09: DR_Swap message.
CTRL_MSG_PR_SWAP	0x0A: PR_Swap message.
CTRL_MSG_VCONN_SWAP	0x0B: VCONN_Swap message.
CTRL_MSG_WAIT	0x0C: Wait message.
CTRL_MSG_SOFT_RESET	0x0D: Soft_Reset message.
CTRL_MSG_NOT_SUPPORTED	0x0E: Not_Supported message.
CTRL_MSG_GET_SRC_CAP_EXTD	0x0F: Get_Source_Cap_Extended message.
CTRL_MSG_GET_STATUS	0x10: Get_Status message .
CTRL_MSG_FR_SWAP	0x11: FR_Swap message.
CTRL_MSG_GET_PPS_STATUS	0x12: Get_PPS_Status message.
CTRL_MSG_GET_COUNTRY_CODES	0x13: Get_Country_Codes message.

6.20.4.10 data_msg_t

enum `data_msg_t`

Enum of the data message types.

Enumerator

DATA_MSG_SRC_CAP	0x01: Source_Capabilities message.
DATA_MSG_REQUEST	0x02: Request message.
DATA_MSG_BIST	0x03: BIST message.
DATA_MSG_SNK_CAP	0x04: Sink_Capabilities message.
DATA_MSG_BAT_STATUS	0x05: Battery_Status message.
DATA_MSG_ALERT	0x06: Alert message.
DATA_MSG_GET_COUNTRY_INFO	0x07: Get_Country_Info message.
DATA_MSG_VDM	0x0F: Vendor_Defined message.

6.20.4.11 dpm_pd_cmd_t

```
enum dpm_pd_cmd_t
```

Enum of the DPM (Device Policy Manager) command types.

Enumerator

DPM_CMD_SRC_CAP_CHNG	Source Caps changed notification. Can be used to trigger fresh contract.
DPM_CMD_SNK_CAP_CHNG	Sink Caps changed notification. Can be used to trigger fresh contract.
DPM_CMD_SEND_GO_TO_MIN	Send GotoMin message to port partner.
DPM_CMD_GET_SNK_CAP	Send Get_Sink_Cap message to port partner.
DPM_CMD_GET_SRC_CAP	Send Get_Source_Cap message to port partner.
DPM_CMD_SEND_HARD_RESET	Send Hard Reset.
DPM_CMD_SEND_SOFT_RESET	Send Soft Reset to port partner.
DPM_CMD_SEND_CABLE_RESET	Send Cable Reset.
DPM_CMD_SEND_SOFT_RESET_EMCA	Send Soft Reset to cable marker.
DPM_CMD_SEND_DR_SWAP	Send DR_Swap request.
DPM_CMD_SEND_PR_SWAP	Send PR_Swap request.
DPM_CMD_SEND_VCONN_SWAP	Send VCONN_Swap request.
DPM_CMD_SEND_VDM	Send VDM message.
DPM_CMD_SEND_EXTENDED	Send extended data message.
DPM_CMD_GET_SRC_CAP_EXTENDED	Send Get_Source_Cap_Extended message.
DPM_CMD_GET_STATUS	Send Get_Status message.
DPM_CMD_SEND_BATT_STATUS	Send Battery_Status data message.
DPM_CMD_SEND_ALERT	Send Alert message.
DPM_CMD_SEND_NOT_SUPPORTED	Send Not_Supported message.
DPM_CMD_SEND_INVALID	Invalid command code.

6.20.4.12 dpm_typec_cmd_resp_t

```
enum dpm_typec_cmd_resp_t
```

Enum of the DPM (Device Policy Manager) response types.

Enumerator

DPM_RESP_FAIL	Command failed.
DPM_RESP_SUCCESS	Command succeeded.

6.20.4.13 dpm_typec_cmd_t

enum [dpm_typec_cmd_t](#)

Enum of the DPM (Device Policy Manager) command types that can be initiated through the `dpm_typec_command` API.

See also

[dpm_typec_command](#)

Enumerator

DPM_CMD_SET_RP_DFLT	Command to select Default Rp.
DPM_CMD_SET_RP_1_5A	Command to select 1.5 A Rp.
DPM_CMD_SET_RP_3A	Command to select 3 A Rp.
DPM_CMD_PORT_DISABLE	Command to disable the USB-PD port.
DPM_CMD_TYPEC_ERR_RECOVERY	Command to initiate Type-C error recovery.
DPM_CMD_TYPEC_INVALID	Invalid command type.

6.20.4.14 extd_msg_t

enum [extd_msg_t](#)

Enum of the extended data message types.

Enumerator

EXTD_MSG_SRC_CAP_EXTD	0x01: Source_Capabilities_Extended message.
EXTD_MSG_STATUS	0x02: Status message.
EXTD_MSG_GET_BAT_CAP	0x03: Get_Battery_Cap message.
EXTD_MSG_GET_BAT_STATUS	0x04: Get_Battery_Status message.
EXTD_MSG_BAT_CAP	0x05: Battery_Capabilities message.
EXTD_MSG_GET_MANU_INFO	0x06: Get_Manufacturer_Info message.
EXTD_MSG_MANU_INFO	0x07: Manufacturer_Info message.
EXTD_MSG_SECURITY_REQ	0x08: Security_Request message.
EXTD_MSG_SECURITY_RESP	0x09: Security_Response message.
EXTD_MSG_FW_UPDATE_REQ	0x0A: Firmware_Update_Request message.
EXTD_MSG_FW_UPDATE_RESP	0x0B: Firmware_Update_Response message.
EXTD_MSG_PPS_STATUS	0x0C: PPS_Status message.
EXTD_MSG_COUNTRY_INFO	0x0D: Country_Info message.
EXTD_MSG_COUNTRY_CODES	0x0E: Country_Codes message.

6.20.4.15 fr_swap_supp_t

```
enum fr_swap_supp_t
```

Enumerator

FR_SWAP_NOT_SUPPORTED	FR_Swap is not supported.
FR_SWAP_DEF_USB	Device will sink less than 900 mA of current after FR_Swap.
FR_SWAP_1_5A	Device will sink less than 1.5 A of current after FR_Swap.
FR_SWAP_3A	Device will sink less than 3 A of current after FR_SWAP.

6.20.4.16 pd_ams_type

```
enum pd_ams_type
```

Enumerator

PD_AMS_NONE	No AMS active.
PD_AMS_NON_INTR	Non-interruptible AMS is active.
PD_AMS_INTR	Interruptible AMS is active.

6.20.4.17 pd_cable_reset_reason_t

```
enum pd_cable_reset_reason_t
```

Enumeration of reasons for issuing a Cable Reset.

Enumerator

EMCA_CABLE_RES_NONE	No Cable Reset performed.
EMCA_CABLE_RES_SR_TIMEOUT	SOP' or SOP" SoftReset timed out.

6.20.4.18 pd_contract_status_t

```
enum pd_contract_status_t
```

Enum of possible PD contract negotiation scenarios that are used to signal the application event handler. This status will be reported in byte 0 of the event data passed along with the APP_EVT_PD_CONTRACT_NEGOTIATION_COMPLETE event. Bytes 3:1 of the event data are not used; and bytes 7:4 will report the RDO where applicable.

Enumerator

PD_CONTRACT_NEGOTIATION_SUCCESSFUL	PD contract negotiation successful.
------------------------------------	-------------------------------------

Enumerator

PD_CONTRACT_CAP_MISMATCH_DETECTED	PD contract negotiated, but capability mismatch is present.
PD_CONTRACT_REJECT_CONTRACT_VALID	Contract rejected by CCG, but previous contract is still valid.
PD_CONTRACT_REJECT_CONTRACT_NOT_VA↔ LID	Contract rejected by CCG and previous contract became invalid.
PD_CONTRACT_REJECT_NO_CONTRACT	Contract rejected by CCG and there was no previous contract.
PD_CONTRACT_REJECT_EXPLICIT_CONTRACT	Request rejected by port partner while in previous explicit contract.
PD_CONTRACT_REJECT_NO_EXPLICIT_CONT↔ RACT	Request rejected by port partner with no previous explicit contract.
PD_CONTRACT_PS_READY_NOT_RECEIVED	Failed to receive PS_RDY after Accept.
PD_CONTRACT_PS_READY_NOT_SENT	Failed to send PS_RDY after Accept.

6.20.4.19 pd_devtype_t

```
enum pd_devtype_t
```

Enum of the attached device type.

Enumerator

DEV_SNK	Power sink device is attached.
DEV_SRC	Power source device is attached.
DEV_DBG_ACC	Debug accessory is attached.
DEV_AUD_ACC	Audio accessory is attached.
DEV_PWRD_ACC	Powered accessory is attached.
DEV_UNSUPORTED_ACC	Unsupported device type is attached.

6.20.4.20 pd_emca_sr_reason_t

```
enum pd_emca_sr_reason_t
```

Enumeration of possible reasons to issue an EMCA (SOP' or SOP") soft reset.

Enumerator

EMCA_SR_REASON_NONE	No EMCA soft-reset in progress.
EMCA_SR_CABLE_DISC	EMCA soft-reset due to cable discovery.
EMCA_SR_ALT_MODE_DISC	EMCA soft-reset due to alt mode discovery.

6.20.4.21 pd_err_recov_reason_t

```
enum pd_err_recov_reason_t
```

Enumeration of reasons for error recovery entry.

Enumerator

ERR_RECov_REASON_NONE	Error recovery is not active.
ERR_RECov_HR_FAIL	Error recovery due to hard reset failure.
ERR_RECov_PROTECT_FAULT	Error recovery due to protection (OVP/OCP) fault.
ERR_RECov_POWER_FAULT	Error recovery due to voltage fault.
ERR_RECov_BAD_DATA_ROLE	Error recovery due to bad data role in incoming PD message.
ERR_RECov_FRS_FAIL	Error recovery due to Fast Role Swap error.

6.20.4.22 pd_hard_reset_reason_t

```
enum pd_hard_reset_reason_t
```

Enumeration of reasons for issuing a hard reset.

Enumerator

PD_HARDRES_REASON_NONE	HardReset not issued.
PD_HARDRES_REASON_NO_SRC_CAP	No Source Capability messages received.
PD_HARDRES_REASON_HOSTCONN	TBT Host Connect state change.
PD_HARDRES_REASON_SR_ERROR	SoftReset failed.
PD_HARDRES_REASON_CONTRACT_ERROR	Power contract failed.
PD_HARDRES_REASON_DR_SWAP	DR Swap received while in Alternate Mode.
PD_HARDRES_REASON_VBUS_OVP	Over-Voltage condition detected.
PD_HARDRES_REASON_VBUS_OCP	Over-Current condition detected.
PD_HARDRES_REASON_AMS_ERROR	PD Atomic Message Sequence error.

6.20.4.23 pd_msg_class_t

```
enum pd_msg_class_t
```

Enumerator

PD_CTRL_MSG	Control message.
PD_DATA_MSG	Data message.
PD_EXTD_MSG	Extended data message.
PD_CABLE_RESET	Cable reset message.
PD_MSG_RSVD	Undefined message type.

6.20.4.24 pd_rev_t

enum `pd_rev_t`

Enumeration of the PD spec revisions.

Enumerator

PD_REV1	USB-PD spec revision 1.0. Not supported.
PD_REV2	USB-PD spec revision 2.0.
PD_REV3	USB-PD spec revision 3.0.
PD_REV_RSVD	Undefined USB-PD spec revision.

6.20.4.25 pd_soft_reset_reason_t

enum `pd_soft_reset_reason_t`

Enumeration of reasons for issuing a soft reset.

Enumerator

PD_SOFTRES_REASON_NONE	SoftReset not issued.
PD_SOFTRES_REASON_SRCNEG_ERROR	Contract negotiation error when CCGx is source.
PD_SOFTRES_REASON_SNKNEG_ERROR	Contract negotiation error when CCGx is sink.
PD_SOFTRES_REASON_AMS_ERROR	PD protocol error.

6.20.4.26 pd_timer_id_t

enum `pd_timer_id_t`

Soft timer ID assignments for various Type-C, USB-PD and Application level periodic tasks. These timer IDs should not be changed; and new timer IDs should be chosen from free ranges.

Please refer to timer definitions in [app.h](#) to avoid conflicts with timer IDs used in default App layer code.

Enumerator

PD_TIMERS_START_ID	000: Start index for USB-PD stack timers.
PD_CABLE_TIMER	001: Timer used for cable capability check.
PD_NO_RESPONSE_TIMER	002: Response timer.
PD_CBL_DSC_ID_TIMER	003: Timer used for cable discovery state machine.
PD_CBL_DELAY_TIMER	004: Timer used to enforce cable delay.
PD_PHY_BUSY_TIMER	005: Timer used to handle PHY busy status.
PD_GOOD_CRC_TX_TIMER	006: GoodCRC timer.
PD_HARD_RESET_TX_TIMER	007: Hard reset transmit timer.
PD_VCONN_SWAP_INITIATOR_TIMER	008: VConn swap initiator timer.
PD_GENERIC_TIMER	009: Generic AMS timer.
PD_PPS_TIMER	010: PPS related timer.
PD_SINK_TX_TIMER	011: PD 3.0 sink Rp flow control timer.
PD_TIMERS_END_ID	014: End index (inclusive) for USB-PD stack timers.

Enumerator

<code>TYPEC_TIMERS_START_ID</code>	015: Start index for Type-C timers.
<code>TYPEC_CC1_DEBOUNCE_TIMER</code>	015: Timer used for CC1 debounce.
<code>TYPEC_CC2_DEBOUNCE_TIMER</code>	016: Timer used for CC2 debounce.
<code>TYPEC_RD_DEBOUNCE_TIMER</code>	017: Timer used for Rd debounce.
<code>TYPEC_VBUS_DISCHARGE_TIMER</code>	018: VBus discharge timer id.
<code>TYPEC_ACTIVITY_TIMER</code>	019: Type-C activity timer id.
<code>TYPEC_GENERIC_TIMER1</code>	020: Generic Type-C state machine timer #1.
<code>TYPEC_GENERIC_TIMER2</code>	021: Generic Type-C state machine timer #2.
<code>TYPEC_TIMERS_END_ID</code>	024: End index (inclusive) for Type-C timers.
<code>PD_OCP_DEBOUNCE_TIMER</code>	025: Timer used for FW debounce of VBus OCP.
<code>HPD_RX_ACTIVITY_TIMER_ID</code>	026: Timer used for HPD receive handling.
<code>PD_VCONN_OCP_DEBOUNCE_TIMER</code>	027: Timer used for FW debounce of VConn OCP.
<code>VBUS_DISCHARGE_SCHEDULE_TIMER</code>	029: Timer used to monitor VBus discharge operation.
<code>APP_TIMERS_START_ID</code>	030: Start index for Application level timers.

6.20.4.27 pdo_t

```
enum pdo_t
```

Enum of the PDO types.

Enumerator

<code>PDO_FIXED_SUPPLY</code>	Fixed (voltage) supply power data object.
<code>PDO_BATTERY</code>	Battery based power data object.
<code>PDO_VARIABLE_SUPPLY</code>	Variable (voltage) supply power data object.
<code>PDO_AUGMENTED</code>	Augmented power data object.

6.20.4.28 pe_cbl_state_t

```
enum pe_cbl_state_t
```

Enum of the Policy Engine cable discovery states.

Enumerator

<code>CBL_FSM_DISABLED</code>	Cable state machine is inactive.
<code>CBL_FSM_ENTRY</code>	Cable state machine starting up.
<code>CBL_FSM_SEND_SOFT_RESET</code>	Cable state machine sending Soft Reset to cable marker.
<code>CBL_FSM_SEND_DSC_ID</code>	Cable state machine waiting for cable response.

6.20.4.29 pe_fsm_state_t

enum `pe_fsm_state_t`

Enumeration of Policy Engine states for a USB-PD port. This is for internal stack usage.

Warning

The ordering of elements must not be altered unless the state table in the stack source is also updated.

Enumerator

<code>PE_FSM_OFF</code>	Policy Engine not started.
<code>PE_FSM_HR_SEND</code>	Send HardReset
<code>PE_FSM_HR_SRC_TRANS_DFLT</code>	<code>PE_SRC_Transition_to_default</code>
<code>PE_FSM_HR_SRC_RECOVER</code>	Policy Engine waiting for recovery before enabling VBus.
<code>PE_FSM_HR_SRC_VBUS_ON</code>	Policy Engine enabling VBus after Hard Reset completion.
<code>PE_FSM_HR_SNK_TRANS_DFLT</code>	<code>PE_SNK_Transition_to_default</code>
<code>PE_FSM_HR_SNK_WAIT_VBUS_OFF</code>	Policy Engine waiting for VBus turning off.
<code>PE_FSM_HR_SNK_WAIT_VBUS_ON</code>	Policy Engine waiting for VBus to turn back on.
<code>PE_FSM_BIST_TEST_DATA</code>	BIST test data state.
<code>PE_FSM_BIST_CM2</code>	<code>PE_BIST_Carrier_Mode</code>
<code>PE_FSM_SNK_STARTUP</code>	<code>PE_SNK_Startup</code>
<code>PE_FSM_SNK_WAIT_FOR_CAP</code>	<code>PE_SNK_Wait_for_Capabilities</code>
<code>PE_FSM_SNK_EVAL_CAP</code>	<code>PE_SNK_Evaluate_Capability</code>
<code>PE_FSM_SNK_SEL_CAP</code>	<code>PE_SNK_Select_Capability</code>
<code>PE_FSM_SRC_STARTUP</code>	<code>PE_SRC_Startup</code>
<code>PE_FSM_SRC_WAIT_NEW_CAP</code>	<code>PE_SRC_Wait_New_Capabilities</code>
<code>PE_FSM_SRC_SEND_CBL_SR</code>	<code>PE_CBL_Soft_Reset</code>
<code>PE_FSM_SRC_SEND_CBL_DSCID</code>	<code>PE_CBL_Get_Identity</code>
<code>PE_FSM_SRC_SEND_CAP</code>	<code>PE_SRC_Send_Capabilities</code>
<code>PE_FSM_SRC_DISCOVERY</code>	<code>PE_SRC_Discovery</code>
<code>PE_FSM_SRC_NEG_CAP</code>	<code>PE_SRC_Negotiate_Capability</code>
<code>PE_FSM_SRC_TRANS_SUPPLY</code>	<code>PE_SRC_Transition_Supply</code>
<code>PE_FSM_SRC_SEND_PS_RDY</code>	Policy Engine waiting to send PS_RDY.
<code>PE_FSM_SNK_TRANS</code>	<code>PE_SNK_Transition_Sink</code>
<code>PE_FSM_SR_SEND</code>	Policy Engine sending Soft Reset.
<code>PE_FSM_SR_RCVD</code>	Policy Engine received Soft Reset.
<code>PE_FSM_VRS_VCONN_ON</code>	Policy Engine waiting for VConn to turn on.
<code>PE_FSM_VRS_VCONN_OFF</code>	Policy Engine waiting for VConn to turn off.
<code>PE_FSM_SWAP_EVAL</code>	Evaluate received swap command.
<code>PE_FSM_SWAP_SEND</code>	Waiting to send swap command.
<code>PE_FSM_DRS_CHANGE_ROLE</code>	Change data role.
<code>PE_FSM_PRS_SRC_SNK_TRANS</code>	Source to Sink PR_Swap transition start.
<code>PE_FSM_PRS_SRC_SNK_VBUS_OFF</code>	Initial source waiting for VBus turning off.
<code>PE_FSM_PRS_SRC_SNK_WAIT_PS_RDY</code>	Initial source waiting for PS_RDY.
<code>PE_FSM_PRS_SNK_SRC_WAIT_PS_RDY</code>	Initial sink waiting for PS_RDY.
<code>PE_FSM_PRS_SNK_SRC_VBUS_ON</code>	Initial sink turning VBus ON.
<code>PE_FSM_FRS_CHECK_RP</code>	Initial sink checking Rp to send FR_Swap message.
<code>PE_FSM_FRS_SRC_SNK_CC_SIGNAL</code>	Initial source sending FR_Swap signal.
<code>PE_FSM_READY</code>	<code>PE_Ready</code> state.

Enumerator

PE_FSM_SEND_MSG	Policy Engine sending new AMS.
PE_FSM_MAX_STATES	Invalid Policy Engine state.

6.20.4.30 peak_cur_cap_t

```
enum peak\_cur\_cap\_t
```

Enum of Peak Current Capability levels.

Enumerator

IMAX_EQ_IOC	Peak current equal to operating current.
IMAX_EQ_130_IOC	Peak current is 1.3x operating current.
IMAX_EQ_150_IOC	Peak current is 1.5x operating current.
IMAX_EQ_200_IOC	Peak current is 2x operating current.

6.20.4.31 port_role_t

```
enum port\_role\_t
```

Enum of the PD port roles.

Enumerator

PRT_ROLE_SINK	Power sink
PRT_ROLE_SOURCE	Power source
PRT_DUAL	Dual Role Power device: can be source or sink.

6.20.4.32 port_type_t

```
enum port\_type\_t
```

Enum of the PD port types.

Enumerator

PRT_TYPE_UFP	Upstream facing port. USB device or Alternate mode accessory.
PRT_TYPE_DFP	Downstream facing port. USB host or Alternate mode controller.
PRT_TYPE_DRP	Dual Role data device: can be UFP or DFP.

6.20.4.33 rd_cc_status_t

enum `rd_cc_status_t`

Enum of the Rd status when Rd is asserted.

Enumerator

RD_RA	CCG has applied Rd. No external Rp.
RD_USB	CCG has applied Rd. Default Rp present.
RD_1_5A	CCG has applied Rd. 1.5A Rp present.
RD_3A	CCG has applied Rd. 3A Rp present.
RD_ERR	CCG has applied Rd. Error state.

6.20.4.34 rdo_type_t

enum `rdo_type_t`

Enum of the RDO types.

Enumerator

FIXED_VAR_RDO	Fixed or variable supply request data object.
BAT_RDO	Battery request data object.

6.20.4.35 resp_status_t

enum `resp_status_t`

Enum of the response status to DPM commands.

Enumerator

SEQ_ABORTED	PD AMS aborted.
CMD_FAILED	PD AMS failed.
RES_TIMEOUT	No response received.
CMD_SENT	PD command has been sent. Response wait may be in progress.
RES_RCVD	Response received.

6.20.4.36 rp_cc_status_t

enum `rp_cc_status_t`

Enum of the Rp status when Rp is asserted.

Enumerator

RP_RA	CCG has applied Rp. External Ra present.
-------	--

Enumerator

RP_RD	CCG has applied Rp. External Rd present.
RP_OPEN	CCG has applied Rp. No external pulldown.

6.20.4.37 rp_term_t

```
enum rp_term_t
```

Enum of the CC termination current levels.

Enumerator

RP_TERM_RP_CUR_DEF	Use default Rp.
RP_TERM_RP_CUR_1_5A	Use 1.5 A Rp.
RP_TERM_RP_CUR_3A	Use 3A Rp.

6.20.4.38 sop_t

```
enum sop_t
```

Enum of the SOP (Start Of Frame) types.

Enumerator

SOP	SOP: Used for communication with port partner.
SOP_PRIME	SOP': Cable marker communication.
SOP_DPRIME	SOP'': Cable marker communication.
SOP_P_DEBUG	SOP'_Debug
SOP_DP_DEBUG	SOP''_Debug
HARD_RESET	Hard Reset
CABLE_RESET	Cable Reset
SOP_INVALID	Undefined ordered set.

6.20.4.39 std_vdm_cmd_t

```
enum std_vdm_cmd_t
```

Enum of the standard VDM commands.

Enumerator

VDM_CMD_DSC_IDENTITY	Discover Identity command.
VDM_CMD_DSC_SVIDS	Discover SVIDs command.
VDM_CMD_DSC_MODES	Discover Modes command.
VDM_CMD_ENTER_MODE	Enter Mode command.
VDM_CMD_EXIT_MODE	Exit Mode command.

Enumerator

VDM_CMD_ATTENTION	Attention message.
VDM_CMD_DP_STATUS_UPDT	DisplayPort Status Update message.
VDM_CMD_DP_CONFIGURE	DisplayPort Configure command.

6.20.4.40 std_vdm_cmd_type_t

```
enum std_vdm_cmd_type_t
```

Enum of the standard VDM command types.

Enumerator

CMD_TYPE_INITIATOR	VDM sent by command initiator.
CMD_TYPE_RESP_ACK	ACK response.
CMD_TYPE_RESP_NAK	NAK response.
CMD_TYPE_RESP_BUSY	BUSY response.

6.20.4.41 std_vdm_prod_t

```
enum std_vdm_prod_t
```

Enum of the standard VDM product types.

Enumerator

PROD_TYPE_UNDEF	Undefined device type.
PROD_TYPE_HUB	Hub device type.
PROD_TYPE_PERI	Peripheral device type.
PROD_TYPE_PAS_CBL	Passive Cable.
PROD_TYPE_ACT_CBL	Active Cable.
PROD_TYPE_AMA	Alternate Mode Accessory.

6.20.4.42 std_vdm_ver_t

```
enum std_vdm_ver_t
```

Enum for the standard VDM version.

Enumerator

STD_VDM_VER1	VDM version 1.0
STD_VDM_VER2	VDM version 2.0
STD_VDM_VER3	VDM version 3.0

Enumerator

STD_VDM_VER4	VDM version 4.0
--------------	--------------------

6.20.4.43 try_src_snk_t

```
enum try_src_snk_t
```

Enum of the Try Source/ Try Sink options.

Enumerator

TRY_SRC_TRY_SNK_DISABLED	Try.SRC and Try.SNK disabled.
TRY_SRC_ENABLED	Try.SRC enabled.
TRY_SNK_ENABLED	Try.SNK enabled.

6.20.4.44 typec_fsm_state_t

```
enum typec_fsm_state_t
```

Enum of the Type-C FSM states. This is for internal stack usage.

Warning

The ordering of elements must not be altered unless the state table is also updated to match.

Enumerator

TYPEC_FSM_DISABLED	Type-C state machine is disabled.
TYPEC_FSM_ERR_RECOV	Error Recovery state.
TYPEC_FSM_ATTACH_WAIT	AttachWait.SRC or AttachWait.SNK state.
TYPEC_FSM_TRY_SRC	Try.SRC state.
TYPEC_FSM_TRY_WAIT_SNK	TryWait.SNK state.
TYPEC_FSM_TRY_SNK	Try.SNK state.
TYPEC_FSM_TRY_WAIT_SRC	TryWait.SRC state.
TYPEC_FSM_UNATTACHED_SRC	Unattached.SRC state.
TYPEC_FSM_UNATTACHED_SNK	Unattached.SNK state.
TYPEC_FSM_AUD_ACC	AudioAccessory state.
TYPEC_FSM_DBG_ACC	DebugAccessory state.
TYPEC_FSM_ATTACHED_SRC	Attached.SRC state.
TYPEC_FSM_ATTACHED_SNK	Attached.SNK state.
TYPEC_FSM_MAX_STATES	Invalid Type-C state.

6.20.4.45 vdm_ams_t

enum `vdm_ams_t`

Enumeration of application responses to policy manager.

Enumerator

<code>VDM_AMS_RESP_READY</code>	Response is ready
<code>VDM_AMS_RESP_NOT_REQ</code>	No response required
<code>VDM_AMS_RESP_FROM_EC</code>	Response will come from EC
<code>VDM_AMS_RESP_NOT_SUPP</code>	Send a NOT_SUPPORTED response.

6.20.4.46 vdm_type_t

enum `vdm_type_t`

Enum of the VDM types.

Enumerator

<code>VDM_TYPE_UNSTRUCTURED</code>	Unstructured VDM.
<code>VDM_TYPE_STRUCTURED</code>	Structured VDM.

6.20.5 Function Documentation

6.20.5.1 get_pd_config()

```
const pd_config_t* get_pd_config (
    void )
```

This function gets the config table data.

Returns

Returns a pointer to the config table info structure.

Warning

The information provided by this API must not be altered by the application.

6.20.5.2 get_pd_port_config()

```
const pd_port_config_t* get_pd_port_config (
    uint8_t port )
```

This function gets the configuration information for the specified port.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

Returns a pointer to the port specific config table info structure.

Warning

The information provided by this API must not be altered by the application.

6.20.5.3 pd_get_ptr_bat_chg_tbl()

```
bat_chg_params_t* pd_get_ptr_bat_chg_tbl (
    uint8_t port )
```

Retrieve pointer to battery power parameters from the configuration table.

This function retrieves the battery charging parameters that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to battery charging parameters.

6.20.5.4 pd_get_ptr_bb_tbl()

```
bb_settings_t* pd_get_ptr_bb_tbl (
    uint8_t port )
```

Retrieve pointer to Billboard settings from the configuration table.

This function retrieves the Billboard settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to Billboard settings.

6.20.5.5 pd_get_ptr_chg_cfg_tbl()

```
chg_cfg_params_t* pd_get_ptr_chg_cfg_tbl (
    uint8_t port )
```

Retrieve pointer to battery charging parameters from the configuration table.

This function retrieves the legacy charging parameters that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to legacy charging parameters.

6.20.5.6 pd_get_ptr_ocp_tbl()

```
ocp_settings_t* pd_get_ptr_ocp_tbl (
    uint8_t port )
```

Retrieve pointer to VBus OCP settings from the configuration table.

This function retrieves the VBus OCP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to VBus OCP settings.

6.20.5.7 pd_get_ptr_otp_tbl()

```
otp_settings_t* pd_get_ptr_otp_tbl (
    uint8_t port )
```

Retrieve pointer to OTP settings from the configuration table.

This function retrieves the OTP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to OTP settings.

6.20.5.8 pd_get_ptr_otp_tbl()

```
otp_settings_t* pd_get_ptr_otp_tbl (
    uint8_t port )
```

Retrieve pointer to VBus OTP settings from the configuration table.

This function retrieves the VBus OTP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to VBus OTP settings.

6.20.5.9 pd_get_ptr_pwr_tbl()

```
pwr_params_t* pd_get_ptr_pwr_tbl (
    uint8_t port )
```

Retrieve pointer to power parameters from the configuration table.

This function retrieves the power parameters that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to power parameters.

6.20.5.10 pd_get_ptr_scp_tbl()

```
scp_settings_t* pd_get_ptr_scp_tbl (
    uint8_t port )
```

Retrieve pointer to VBus SCP settings from the configuration table.

This function retrieves the VBus SCP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to VBus SCP settings.

6.20.5.11 pd_get_ptr_type_a_chg_cfg_tbl()

```
typeA_chg_cfg_params_t* pd_get_ptr_type_a_chg_cfg_tbl (
    uint8_t port )
```

Retrieve pointer to TYPE-A battery charging parameters from the configuration table.

This function retrieves the legacy charging parameters that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to legacy charging parameters.

6.20.5.12 pd_get_ptr_type_a_pwr_tbl()

```
pwr_params_t* pd_get_ptr_type_a_pwr_tbl (
    uint8_t port )
```

Retrieve pointer to power parameters of Type-A port from the configuration table.

This function retrieves the power parameters of Type-A port that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to power parameters of Type-A port.

6.20.5.13 pd_get_ptr_uvp_tbl()

```
uvp_settings_t* pd_get_ptr_uvp_tbl (
    uint8_t port )
```

Retrieve pointer to VBus UVP settings from the configuration table.

This function retrieves the VBus UVP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to VBus UVP settings.

6.20.5.14 pd_get_ptr_vconn_ocp_tbl()

```
vconn_ocp_settings_t* pd_get_ptr_vconn_ocp_tbl (
    uint8_t port )
```

Retrieve pointer to Vconn OCP settings from the configuration table.

This function retrieves the Vconn OCP settings that is stored in the configuration table and stores it in the run-time data structures.

Parameters

<i>port</i>	USB-PD port for which the data is to be retrieved.
-------------	--

Returns

Pointer to Vconn OCP settings.

6.20.5.15 pd_is_msg()

```
bool pd_is_msg (
    const pd_packet_t * pkt,
    sop_t sop_type,
    pd_msg_class_t msg_class,
    uint32_t msg_mask,
    uint16_t length )
```

This is a utility function to check if the packet is an expected message of a certain class.

Parameters

<i>pkt</i>	Packet pointer.
<i>sop_type</i>	Sop type to match. Passing SOP_INVALID will skip this check.
<i>msg_class</i>	Message class: Control, Data or Extended.
<i>msg_mask</i>	Message mask is a 32 bit mask. Each bit corresponds to a message type corresponding to the message class. If mask is 0, then this check is skipped.
<i>length</i>	Length corresponds to the 'Number of Data Objects' field in the PD header. If length is 0xFFFF, this check is skipped.

Returns

Returns true if packet matches all the conditions, else false.

6.21 pd_common/pd_policy_engine.h File Reference

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

Functions

- void `pe_init` (uint8_t port)
- void `pe_fsm` (uint8_t port)
- void `pe_start` (uint8_t port)
- void `pe_stop` (uint8_t port)
- bool `pe_is_busy` (uint8_t port)
- void `pe_clear_hard_reset_count` (uint8_t port)
- bool `get_spec_rev_determined` (uint8_t port)
- void `pe_disabled` (uint8_t port)
- void `pe_push_to_buf` (uint8_t port, uint8_t state)
- uint8_t * `get_pe_state_buf` (uint8_t port)
- void `pe_get_pps_status` (uint8_t port, uint8_t *pps_status)

6.21.1 Detailed Description

USB-PD policy engine header file.

6.21.2 Function Documentation

6.21.2.1 `get_pe_state_buf()`

```
uint8_t* get_pe_state_buf (
    uint8_t port )
```

This function returns the Policy Engine FSM history buffer.

Parameters

<i>port</i>	port index
-------------	------------

6.21.2.2 `get_spec_rev_determined()`

```
bool get_spec_rev_determined (
    uint8_t port )
```

Checks if spec rev bit in dpm_status register is valid.

Parameters

<i>port</i>	port index
-------------	------------

Returns

Returns true if spec rev is determined else return false

6.21.2.3 pe_clear_hard_reset_count()

```
void pe_clear_hard_reset_count (
    uint8_t port )
```

Clears hard reset count.

Parameters

<i>port</i>	port index
-------------	------------

Returns

None

6.21.2.4 pe_disabled()

```
void pe_disabled (
    uint8_t port )
```

Disables the policy engine. PD port is limited to receiving hard resets.

Parameters

<i>port</i>	port index
-------------	------------

6.21.2.5 pe_fsm()

```
void pe_fsm (
    uint8_t port )
```

This function runs pe state machine.

Parameters

<i>port</i>	port index
-------------	------------

6.21.2.6 pe_get_pps_status()

```
void pe_get_pps_status (
    uint8_t port,
    uint8_t * pps_status )
```

This function returns the PPS status.

Parameters

<i>port</i>	port index
<i>pps_status</i>	PPS status data as defined by USB-PD specification.

6.21.2.7 pe_init()

```
void pe_init (
    uint8_t port )
```

This function initializes the policy engine. This function also initializes the pd protocol module.

Parameters

<i>port</i>	port index
-------------	------------

6.21.2.8 pe_is_busy()

```
bool pe_is_busy (
    uint8_t port )
```

Checks if policy engine is busy in some task.

Parameters

<i>port</i>	port index
-------------	------------

Returns

Returns true if busy else return false

6.21.2.9 pe_push_to_buf()

```
void pe_push_to_buf (
    uint8_t port,
    uint8_t state )
```

This function pushes current Policy Engine state to the FSM history buffer.

Parameters

<i>port</i>	port index
<i>state</i>	Next state to be pushed into the buffer.

6.21.2.10 pe_start()

```
void pe_start (
    uint8_t port )
```

This function initialize the internal variables of policy engine to known state and start the pd phy.

Parameters

<i>port</i>	port index
-------------	------------

6.21.2.11 pe_stop()

```
void pe_stop (
    uint8_t port )
```

Completely stops the policy engine. This will stop pd phy as well.

Parameters

<i>port</i>	port index
-------------	------------

6.22 pd_common/pd_protocol.h File Reference

```
#include <pd.h>
#include <status.h>
```

Functions

- [ccg_status_t pd_prot_init \(uint8_t port, pd_cbk_t cbk\)](#)
- [ccg_status_t pd_prot_start \(uint8_t port\)](#)
- [ccg_status_t pd_prot_refresh_roles \(uint8_t port\)](#)
- [ccg_status_t pd_prot_stop \(uint8_t port\)](#)
- [ccg_status_t pd_prot_rx_en \(uint8_t port\)](#)
- [ccg_status_t pd_prot_rx_dis \(uint8_t port, uint8_t hard_reset_en\)](#)
- [bool pd_prot_is_busy \(uint8_t port\)](#)
- [ccg_status_t pd_prot_reset_all \(uint8_t port\)](#)
- [ccg_status_t pd_prot_reset \(uint8_t port, sop_t sop\)](#)
- [ccg_status_t pd_prot_reset_rx \(uint8_t port, sop_t sop\)](#)
- [ccg_status_t pd_prot_send_ctrl_msg \(uint8_t port, sop_t sop, ctrl_msg_t msg_type\)](#)
- [ccg_status_t pd_prot_send_data_msg \(uint8_t port, sop_t sop, data_msg_t msg_type, uint8_t count, pd_do_t *dobj\)](#)
- [ccg_status_t pd_prot_send_hard_reset \(uint8_t port\)](#)
- [ccg_status_t pd_prot_send_cable_reset \(uint8_t port\)](#)
- [ccg_status_t pd_prot_en_bist_cm2 \(uint8_t port\)](#)
- [ccg_status_t pd_prot_dis_bist_cm2 \(uint8_t port\)](#)
- [ccg_status_t pd_prot_en_bist_test_data \(uint8_t port\)](#)
- [ccg_status_t pd_prot_dis_bist_test_data \(uint8_t port\)](#)
- [ccg_status_t pd_prot_set_avoid_retry \(uint8_t port\)](#)
- [pd_packet_extd_t * pd_prot_get_rx_packet \(uint8_t port\)](#)

6.22.1 Detailed Description

USB-PD protocol layer header file.

6.22.2 Function Documentation

6.22.2.1 pd_prot_dis_bist_cm2()

```
ccg_status_t pd_prot_dis_bist_cm2 (
    uint8_t port )
```

This function disable sending bist carrier mode 2. There is no call back for this function This function returns after registering the request.

Parameters

<i>port</i>	port index
-------------	------------

Returns

ccg_status_t

6.22.2.2 pd_prot_dis_bist_test_data()

```
ccg_status_t pd_prot_dis_bist_test_data (
    uint8_t port )
```

This function disables the bist test data mode.

Parameters

<i>port</i>	port index
-------------	------------

Returns

ccg_status_t

6.22.2.3 pd_prot_en_bist_cm2()

```
ccg_status_t pd_prot_en_bist_cm2 (
    uint8_t port )
```

This function enables sending bist carrier mode 2. There is no call back for this function This function returns after registering the request.

Parameters

<i>port</i>	port index
-------------	------------

Returns

<code>ccg_status_t</code>

6.22.2.4 pd_prot_en_bist_test_data()

```
ccg_status_t pd_prot_en_bist_test_data (
    uint8_t port )
```

This function puts the receiver in bist test data mode.

Parameters

<code>port</code>	port index
-------------------	------------

Returns

<code>ccg_status_t</code>

6.22.2.5 pd_prot_get_rx_packet()

```
pd_packet_extd_t* pd_prot_get_rx_packet (
    uint8_t port )
```

This function returns pointer to received pd packet.

Parameters

<code>port</code>	port index
-------------------	------------

Returns

Returns pointer to received data if port param is not correct return null

6.22.2.6 pd_prot_init()

```
ccg_status_t pd_prot_init (
    uint8_t port,
    pd_cbk_t cbk )
```

This function sets the clock and necessary registers for PD hw ip to function and initializes the PD protocol layer.

Parameters

<code>port</code>	port index
<code>cbk</code>	pd event handler callback

Returns

`ccg_status_t`

6.22.2.7 pd_prot_is_busy()

```
bool pd_prot_is_busy (
    uint8_t port )
```

This function checks if protocol layer is busy.

Parameters

<i>port</i>	port index
-------------	------------

Returns

Return true when busy else return false

6.22.2.8 pd_prot_refresh_roles()

```
ccg_status_t pd_prot_refresh_roles (
    uint8_t port )
```

This function configures pd phy as per current port role /data role/contract status of port. This API does not enable the receiver.

Parameters

<i>port</i>	port index
-------------	------------

Returns

`ccg_status_t`

6.22.2.9 pd_prot_reset()

```
ccg_status_t pd_prot_reset (
    uint8_t port,
    sop_t sop )
```

This function resets protocol layer(TX and RX) counter for a specific sop type.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type

Returns

```
ccg_status_t
```

6.22.2.10 pd_prot_reset_all()

```
ccg_status_t pd_prot_reset_all (
    uint8_t port )
```

This function resets the protocol layer(TX and RX) counters for each sop type.

Parameters

<i>port</i>	port index
-------------	------------

Returns

```
ccg_status_t
```

6.22.2.11 pd_prot_reset_rx()

```
ccg_status_t pd_prot_reset_rx (
    uint8_t port,
    sop_t sop )
```

This function resets protocol layer RX only counter for a specific sop type.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type

Returns

```
ccg_status_t
```

6.22.2.12 pd_prot_rx_dis()

```
ccg_status_t pd_prot_rx_dis (
    uint8_t port,
    uint8_t hard_reset_en )
```

This function disables the bmc receiver.

Parameters

<i>port</i>	port index
<i>hard_reset_en</i>	When 0 means rx is completely disabled, When 1 means only hard reset can be received.

Returns

```
ccg_status_t
```

6.22.2.13 pd_prot_rx_en()

```
ccg_status_t pd_prot_rx_en (
    uint8_t port )
```

This function enables the bmc receiver.

Parameters

<i>port</i>	port index
-------------	------------

Returns

```
ccg_status_t
```

6.22.2.14 pd_prot_send_cable_reset()

```
ccg_status_t pd_prot_send_cable_reset (
    uint8_t port )
```

This function sends a cable reset. Results will be known to caller via callback function registered in [pd_phy_init\(\)](#) if this function returns success. This function returns after registering the request.

Parameters

<i>port</i>	port index
-------------	------------

Returns

```
ccg_status_t
```

6.22.2.15 pd_prot_send_ctrl_msg()

```
ccg_status_t pd_prot_send_ctrl_msg (
    uint8_t port,
    sop_t sop,
    ctrl_msg_t msg_type )
```

This function sends a control message. Results will be known to caller via callback function registered in [pd_prot_int\(\)](#) if this function returns success. This function returns after registering the request.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type
<i>msg_type</i>	control message type.

Returns

```
ccg_status_t
```

6.22.2.16 pd_prot_send_data_msg()

```
ccg_status_t pd_prot_send_data_msg (
    uint8_t port,
    sop_t sop,
    data_msg_t msg_type,
    uint8_t count,
    pd_do_t * dobj )
```

This function sends a data message. Results will be known to caller via callback function registered in [pd_phy_init\(\)](#) if this function returns success. This function returns after registering the request.

Parameters

<i>port</i>	port index
<i>sop</i>	sop type
<i>msg_type</i>	data message type
<i>count</i>	data objects count
<i>dobj</i>	pointer to data objects

Returns

```
ccg_status_t
```

6.22.2.17 pd_prot_send_hard_reset()

```
ccg_status_t pd_prot_send_hard_reset (
    uint8_t port )
```

This function sends a hard reset. Results will be known to caller via callback function registered in [pd_phy_init\(\)](#) if this function returns success. This function returns after registering the request.

Parameters

<i>port</i>	port index
-------------	------------

Returns

```
ccg_status_t
```

6.22.2.18 pd_prot_set_avoid_retry()

```
ccg_status_t pd_prot_set_avoid_retry (
    uint8_t port )
```

This function can be used by higher layers to avoid retry on CRC expire for a particular message. When this flag is set before calling `pd_prot_send_data_msg()` or `pd_send_crtl_msg()` APIs, retry is avoided on CRC failure. This is one time only. Flag is automatically cleared after transmission(success or fail).

Parameters

<i>port</i>	port index
-------------	------------

Returns

`ccg_status_t`

6.22.2.19 `pd_prot_start()`

```
ccg_status_t pd_prot_start (
    uint8_t port )
```

This function starts the protocol layer and configures pd phy as per current port role /data role/contract status of port. This API does not enable the receiver.

Parameters

<i>port</i>	port index
-------------	------------

Returns

`ccg_status_t`

6.22.2.20 `pd_prot_stop()`

```
ccg_status_t pd_prot_stop (
    uint8_t port )
```

This function completely stops the pd hw and put it in lowest power state.

Parameters

<i>port</i>	port index
-------------	------------

Returns

`ccg_status_t`

6.23 `pd_common/typec_manager.h` File Reference

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

Functions

- void `typec_init` (uint8_t port)
- void `typec_start` (uint8_t port)
- void `typec_stop` (uint8_t port)
- bool `typec_is_busy` (uint8_t port)
- void `typec_sync_toggle` (void)
- void `typec_deepsleep` (uint8_t port)
- void `typec_wakeup` (void)
- void `typec_fsm` (uint8_t port)
- void `typec_assert_rp` (uint8_t port, uint8_t channel)
- void `typec_assert_rd` (uint8_t port, uint8_t channel)
- void `typec_change_rp` (uint8_t port, `rp_term_t` rp)

6.23.1 Detailed Description

Type-C manager header file.

6.23.2 Function Documentation

6.23.2.1 typec_assert_rd()

```
void typec_assert_rd (
    uint8_t port,
    uint8_t channel )
```

This function asserts Rd and deasserts Rp for the specified CC line.

Parameters

<i>port</i>	Port index.
<i>channel</i>	CC line.

6.23.2.2 typec_assert_rp()

```
void typec_assert_rp (
    uint8_t port,
    uint8_t channel )
```

This function asserts Rp and deasserts Rd for the specified CC line.

Parameters

<i>port</i>	Port index.
<i>channel</i>	CC line.

6.23.2.3 typec_change_rp()

```
void typec_change_rp (
    uint8_t port,
    rp_term_t rp )
```

This function changes Rp. If port is in connected state Rp will changed only on active channel. If port is not connected then Rp get updated on both cc channels.

Parameters

<i>port</i>	Port index.
<i>rp</i>	Desired Rp value.

6.23.2.4 typec_deepsleep()

```
void typec_deepsleep (
    uint8_t port )
```

This function configures Type C functionality in the PD block when entering deepsleep.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.23.2.5 typec_fsm()

```
void typec_fsm (
    uint8_t port )
```

This function implements the Type C state machine.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.23.2.6 typec_init()

```
void typec_init (
    uint8_t port )
```

This function initializes the Type C manager. This function must be called once during system init.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.23.2.7 typec_is_busy()

```
bool typec_is_busy (
    uint8_t port )
```

This function checks if the Type C FSM is busy.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

Returns true if busy, false otherwise.

6.23.2.8 typec_start()

```
void typec_start (
    uint8_t port )
```

This function starts the Type C functionality in the PD hardware block.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.23.2.9 typec_stop()

```
void typec_stop (
    uint8_t port )
```

This function completely disables the Type C FSM and the Type C functionality in the PD hardware block.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.23.2.10 typec_wakeup()

```
void typec_wakeup (
    void )
```

This function configures Type C functionality in the PD block when coming out of deepsleep.

Parameters

<i>port</i>	Port index.
-------------	-------------

6.24 pd_hal/hal_ccgx.h File Reference

```
#include "config.h"
#include "pd.h"
#include "stdint.h"
#include "stdbool.h"
```

Macros

- #define CCG_SILICON_REV00_VALUE (0x11)

Typedefs

- typedef void(* vbus_ocp_cbk_t) (uint8_t port)

Functions

- void system_init (void)
- uint8_t system_vbus_ocp_en (uint8_t port, uint32_t cur, vbus_ocp_cbk_t cbk)
- uint8_t system_vbus_ocp_dis (uint8_t port)
- void vbus_ocp_handler (uint8_t port)
- void system_disconnect_ovp_trip (uint8_t port)
- void system_connect_ovp_trip (uint8_t port)
- uint8_t system_vbus_scp_en (uint8_t port, uint32_t cur, vbus_ocp_cbk_t cbk)
- uint8_t system_vbus_scp_dis (uint8_t port)
- void vbus_scp_handler (uint8_t port)
- uint8_t ccg_get_si_revision (void)

Variables

- const uint8_t csa_tab []

6.24.1 Detailed Description

PD and Type-C HAL layer for CCGx device family.

6.24.2 Macro Definition Documentation

6.24.2.1 CCG_SILICON_REV00_VALUE

```
#define CCG_SILICON_REV00_VALUE (0x11)
< Revision id corresponding to Rev ** version of any CCG silicon.
```

6.24.3 Function Documentation

6.24.3.1 ccg_get_si_revision()

```
uint8_t ccg_get_si_revision (
    void )
```

Function to check CCG silicon revision.

Returns

Returns the silicon revision of the CCG part. 0 for Rev ** and non-zero for others.

6.24.3.2 system_connect_ovp_trip()

```
void system_connect_ovp_trip (
    uint8_t port )
```

This function connects the OVP comparator output to OVP Trip GPIO.

Parameters

<i>port</i>	USB-PD port
-------------	-------------

Returns

None

6.24.3.3 system_disconnect_ovp_trip()

```
void system_disconnect_ovp_trip (
    uint8_t port )
```

This function disconnects the OVP comparator output from OVP Trip GPIO Before disconnecting this function also make sure OVP Trip GPIO is driving strong the last output of comparator. This is useful in using the same comparator for other level comparision.

Parameters

<i>port</i>	USB-PD port
-------------	-------------

Returns

None

6.24.3.4 system_init()

```
void system_init (
    void )
```

USB-PD system initialization.

This function initializes the PD block clocks by setting the divider values for PERI registers and enabling the corresponding control registers.

6.24.3.5 system_vbus_ocp_dis()

```
uint8_t system_vbus_ocp_dis (
    uint8_t port )
```

Disables VBus OCP checks on the specified port.

Parameters

<i>port</i>	USB-PD port on which to disable VBus OCP checks.
-------------	--

Returns

Returns 1 if params are ok else return 0

6.24.3.6 system_vbus_ocp_en()

```
uint8_t system_vbus_ocp_en (
    uint8_t port,
    uint32_t cur,
    vbus_ocp_cbk_t cbk )
```

Enables VBus OCP checks on the specified port.

Parameters

<i>port</i>	USB-PD port on which to enable VBus OCP checks.
<i>cbk</i>	Function to be called when OCP event is detected.

Returns

Returns 1 if params are ok else return 0

6.24.3.7 system_vbus_scp_dis()

```
uint8_t system_vbus_scp_dis (
    uint8_t port )
```

Disables VBus SCP checks on the specified port.

Parameters

<i>port</i>	USB-PD port on which to disable VBus SCP checks.
-------------	--

Returns

Returns 1 if params are ok else return 0

6.24.3.8 system_vbus_scp_en()

```
uint8_t system_vbus_scp_en (
    uint8_t port,
    uint32_t cur,
    vbus_ocp_cbk_t cbk )
```

Enables VBus SCP checks on the specified port.

Parameters

<i>port</i>	USB-PD port on which to enable VBus SCP checks. Negotiated current in 10mA units.
<i>cbk</i>	Function to be called when SCP event is detected.

Returns

Returns 1 if params are ok else return 0

6.24.3.9 vbus_ocp_handler()

```
void vbus_ocp_handler (
    uint8_t port )
```

Vbus OCP interrupt handler.

Parameters

<i>port</i>	USB-PD port on which the OCP interrupt was triggered.
-------------	---

Returns

None

6.24.3.10 vbus_scp_handler()

```
void vbus_scp_handler (
    uint8_t port )
```

Vbus SCP interrupt handler.

Parameters

<i>port</i>	USB-PD port on which the SCP interrupt was triggered.
-------------	---

Returns

None

6.25 pd_hal/hpd.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include <ccgx_regs.h>
#include <status.h>
```

Macros

- #define **HPD_IP_DIRECTION** (0)
- #define **HPD_OP_DIRECTION** (PDSS_CTRL_HPD_DIRECTION)
- #define **HPD_EVENT_MASK** (3u)
- #define **HPD_EVENT_0_POS** (0)
- #define **HPD_EVENT_1_POS** (2)
- #define **HPD_EVENT_2_POS** (4)
- #define **HPD_EVENT_3_POS** (6)
- #define **HPD_GET_EVENT_0**(hpdu_queue) (((hpdu_queue) >> **HPD_EVENT_0_POS**) & **HPD_EVENT_MASK**)
- #define **HPD_GET_EVENT_1**(hpdu_queue) (((hpdu_queue) >> **HPD_EVENT_1_POS**) & **HPD_EVENT_MASK**)
- #define **HPD_GET_EVENT_2**(hpdu_queue) (((hpdu_queue) >> **HPD_EVENT_2_POS**) & **HPD_EVENT_MASK**)
- #define **HPD_GET_EVENT_3**(hpdu_queue) (((hpdu_queue) >> **HPD_EVENT_3_POS**) & **HPD_EVENT_MASK**)

Typedefs

- typedef void(* **hpdu_event_cbk_t**) (uint8_t port, **hpdu_event_type_t** event)

Enumerations

- enum **hpdu_event_type_t** {
 HPD_EVENT_NONE = 0, **HPD_EVENT_UNPLUG**, **HPD_EVENT_PLUG**, **HPD_EVENT_IRQ**,
HPD_COMMAND_DONE }

Functions

- **ccg_status_t hpdu_receive_init** (uint8_t port, **hpdu_event_cbk_t** cbk)
- **ccg_status_t hpdu_transmit_init** (uint8_t port, **hpdu_event_cbk_t** cbk)
- **ccg_status_t hpdu_deinit** (uint8_t port)
- bool **hpdu_receive_get_status** (uint8_t port)
- **ccg_status_t hpdu_transmit_sendevt** (uint8_t port, **hpdu_event_type_t** evtype, bool wait)
- void **hpdu_sleep_entry** (uint8_t port)
- void **hpdu_wakeup** (uint8_t port, bool value)
- void **hpdu_rx_sleep_entry** (uint8_t port, bool hpdu_state)
- void **hpdu_rx_wakeup** (uint8_t port)
- bool **is_hpdu_rx_state_idle** (uint8_t port)

6.25.1 Detailed Description

Hotplug Detect (HPD) driver header file.

6.25.2 Enumeration Type Documentation

6.25.2.1 hpd_event_type_t

```
enum hpd_event_type_t
```

List of HPD events detected by USBPD block. The UNPLUG, PLUG and IRQ events are used in the case of a DisplayPort Sink implementation, and the COMMAND_DONE event is used in the case of a DP Source implementation.

Enumerator

HPD_EVENT_NONE	No event.
HPD_EVENT_UNPLUG	DP Unplug event.
HPD_EVENT_PLUG	DP Plug event.
HPD_EVENT_IRQ	DP IRQ event.
HPD_COMMAND_DONE	Requested HPD command is complete.

6.25.3 Function Documentation

6.25.3.1 hpd_deinit()

```
ccg_status_t hpd_deinit (
    uint8_t port )
```

Disable the HPD functionality on the specified PD port. This is used for HPD receive as well transmit de-initialization.

Parameters

<i>port</i>	PD port index. Caller should ensure to provide only valid values.
-------------	---

Returns

Return CCG_STAT_SUCCESS in case of success, error code otherwise.

6.25.3.2 hpd_receive_get_status()

```
bool hpd_receive_get_status (
    uint8_t port )
```

Returns the current status of the HPD input signal. This function can only be used if the HPD receive block has been enabled using the hpd_receive_init function.

Parameters

<i>port</i>	pd port index.
-------------	----------------

Returns

Current HPD input signal status.

6.25.3.3 hpd_receive_init()

```
ccg_status_t hpd_receive_init (
    uint8_t port,
    hpd_event_cbk_t cbk )
```

Enable the HPD-Receive functionality for the specified PD port. This function shall be used in DP Sink designs and should not be combined with the hpd_transmit_init call.

Parameters

<i>port</i>	PD port index. Caller should ensure to provide only valid values.
<i>cbk</i>	callback function to be called when there is an HPD event.

Returns

Returns CCG_STAT_SUCCESS in case of success, error code otherwise.

6.25.3.4 hpd_rx_sleep_entry()

```
void hpd_rx_sleep_entry (
    uint8_t port,
    bool hpd_state )
```

Prepare the HPD RX block for deep-sleep.

This routine implements a firmware workaround to retain HPD signal state before entering deep sleep. All details of the workaround are captured in function definition.

Parameters

<i>port</i>	Port whose HPD RX is active. Caller should ensure to provide only valid values.
<i>hpd_state</i>	Connection status of HPD return None

6.25.3.5 hpd_rx_wakeup()

```
void hpd_rx_wakeup (
    uint8_t port )
```

Restore the HPD RX block function after CCG wakes up from deep sleep. Implements the wakeup portion of the work-around to retain HPD signal state.

Parameters

<i>port</i>	Port whose HPD RX module settings have to be restored.
-------------	--

Returns

None

6.25.3.6 hpd_sleep_entry()

```
void hpd_sleep_entry (
    uint8_t port )
```

Prepare the HPD transmit block for deep-sleep. This function changes the IO mode of the HPD signal to GPIO, so as to avoid glitches on the signal during the active - sleep - active power mode transitions of the CCG device.

Parameters

<i>port</i>	Port whose HPD output is to be updated. return None
-------------	---

6.25.3.7 hpd_transmit_init()

```
ccg_status_t hpd_transmit_init (
    uint8_t port,
    hpd_event_cbk_t cbk )
```

Enable the HPD-Transmit functionality for the specified PD port. This function shall be used in DP source designs and should not be combined with the hpd_receive_init call.

Parameters

<i>port</i>	PD port index. Caller should ensure to provide only valid values.
<i>cbk</i>	callback to be used for command completion event.

Returns

Returns CCG_STAT_SUCCESS in case of success, error code otherwise.

6.25.3.8 hpd_transmit_sendevt()

```
ccg_status_t hpd_transmit_sendevt (
    uint8_t port,
    hpd_event_type_t evtype,
    bool wait )
```

Send the desired HPD event out through the HPD GPIO. Only the HPD_EVENT_UNPLUG, HPD_EVENT_UNPLUG and HPD_EVENT_IRQ events should be requested.

Parameters

<i>port</i>	Port on which HPD event is to be sent.
<i>evtype</i>	Type of HPD event to be sent.
<i>wait</i>	Whether function should wait until command is complete.

Returns

Returns CCG_STAT_SUCCESS in case of success, error code otherwise.

6.25.3.9 hpd_wakeup()

```
void hpd_wakeup (
    uint8_t port,
    bool value )
```

Restore the HPD Tx signal driver after CCG wakes from deep-sleep.

Parameters

<i>port</i>	Port whose HPD signal is to be restored.
<i>value</i>	Startup value for the HPD signal. The APP layer is expected to identify the desired (PLUG/UNPLUG) state of the signal and pass it to the HAL.

Returns

None

6.25.3.10 is_hpd_rx_state_idle()

```
bool is_hpd_rx_state_idle (
    uint8_t port )
```

Returns the state of HPD RX activity timer. This check should be used to prevent device entry into a low power sleep state while an HPD transition is in progress.

Parameters

<i>port</i>	Port whose HPD RX state is to be checked.
-------------	---

Returns

Status of timer. true if timer is active, false otherwise.

6.26 pd_hal/pdss_hal.h File Reference

```
#include <ccgx_regs.h>
#include <pd.h>
#include <status.h>
```

Macros

- #define **PDSS_DRP_TOGGLE_PERIOD_MS** (75u) /* DRP toggle period in ms. */
- #define **PDSS_DRP_HIGH_PERIOD_MS** (30u) /* Rp assert period in ms. */
- #define **PDSS_DRP_TOGGLE_PERIOD_VAL** (2500u) /* Based on LF clock */

- #define **PDSS_DRP_HIGH_PERIOD_VAL** (1100u) /* Based on LF clock */
- #define **PDSS_CC_filt_CYCLES** (10) /* Based on LF clock */
- #define **PDSS_DP_DM_filt_CYCLES** (10) /* Based on LF clock */
- #define **PDSS_VBUS_IN_REF_SEL_VAL** (1u) /* 500mV */
- #define **SILICON_TRIM6_V1P575_TRIM_VALUE** (3u)
- #define **SILICON_TRIM3_V0P55_TRIM_VALUE** (2u)
- #define **TRIM0_TX_TRIM_VALUE_3A** (2u)
- #define **TRIM0_TX_TRIM_VALUE_900MA** (0u)
- #define **LF_CLK_CYCLE_US** (31u)
- #define **PGDO_PD_ISINK_TERMINAL_VAL** (0x3F)
- #define **AUTO_CTRL_MESSAGE_GOODCRC_MASK_CFG** (0xFFFFFFFFDu)
- #define **AUTO_DATA_MESSAGE_GOODCRC_MASK_CFG** (0xFFFFFFFFFu)
- #define **AUTO_EXTD_MESSAGE_GOODCRC_MASK_CFG** (0xFFFFFFFFFu)
- #define **PD_MSG_HDR_REV2_IGNORE_MASK** (0x8010)
- #define **EXPECTED_GOOD_CRC_HDR_MASK** (0x7E0Fu)
- #define **EXPECTED_GOOD_CRC_HDR_DIFF_MASK_REV3** (0x8010u)
- #define **EXPECTED_GOOD_CRC_CLEAR_MASK** (0xF1FFu)
- #define **RX_CNT_MAX_VAL** (0xFu)
- #define **RX_UI_BOUNDARY_DELTA_VAL** (0x2u)
- #define **RX_CNT_MAX_CFG** (RX_CNT_MAX_VAL << PDSS_RX_CC_0_CFG_RX_CNT_MAX_POS)
- #define **RX_UI_BOUNDARY_DELTA_CFG** (RX_UI_BOUNDARY_DELTA_VAL << PDSS_RX_CC_0_CFG_RX_UI_BOUNDARY_DELTA_POS)
- #define **RX_CC_CFG**
- #define **RX_ORDER_SET_CTRL_CFG**
- #define **CRC_COUNTER_CFG** (300u)
- #define **BUS_IDLE_CNT_VAL** (10u)
- #define **IDLE_COUNTER_VAL** (40u)
- #define **IFG_COUNTER_VAL** (10u)
- #define **BUS_IDLE_CNT_CFG** (BUS_IDLE_CNT_VAL << PDSS_INTER_PACKET_COUNTER_BUS_IDLE_CNT_POS)
- #define **IDLE_COUNTER_CFG** (IDLE_COUNTER_VAL << PDSS_INTER_PACKET_COUNTER_IDLE_COUNTER_POS)
- #define **IFG_COUNTER_CFG** (IFG_COUNTER_VAL << PDSS_INTER_PACKET_COUNTER_IFG_COUNTER_POS)
- #define **INTER_PACKET_COUNTER_CFG** (BUS_IDLE_CNT_CFG | IDLE_COUNTER_CFG | IFG_COUNTER_CFG)
- #define **EXTENDED_DATA_BIT_LOC_VAL** (15u)
- #define **EXTENDED_DATA_BYTE_FIRST_BIT_LOCATION_VAL** (16u)
- #define **EXTENDED_DATA_BYTE_LAST_BIT_LOCATION_VAL** (24u)
- #define **CHUNK_BIT_LOCATION_VAL** (15u)
- #define **HEADER_INFO_CFG**
- #define **PDSS_MAX_TX_MEM_SIZE** (16u)
- #define **PDSS_MAX_TX_MEM_HALF_SIZE** (8u)
- #define **PDSS_MAX_RX_MEM_SIZE** (16u)
- #define **PDSS_MAX_RX_MEM_HALF_SIZE** (8u)
- #define **PD_INTR_MASK** (0x3FF7FBFFu)
- #define **RX_INTERRUPTS**
- #define **RCV_INTR_MASK**
- #define **TX_INTERRUPTS**
- #define **RST_TX_INTERRUPTS**
- #define **EN_DEFAULT_SOP_DET_VAL** (1u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_POS)
- #define **EN_PRIME_SOP_DET_VAL** (2u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_POS)
- #define **EN_DBL_SOP_DET_VAL** (4u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_POS)
- #define **EN_DEBUG_PRIME_DET_VAL** (8u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_POS)
- #define **EN_DEBUG_DBL_DET_VAL** (16u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_POS)

- #define **EN_RX_CABLE_RESET_DET_VAL** (32u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_↔ POS)
- #define **EN_RX_HARD_RESET_DET_VAL** (64u << PDSS_RX_ORDER_SET_CTRL_SOP_RST_EN_↔ POS)
- #define **PDSS_CC_CTRL_0_CMP_LA_VSEL_CFG** (7u)
- #define **PD_CMP_VSEL_0_2_V** (0u)
- #define **PD_CMP_VSEL_0_4_V** (1u)
- #define **PD_CMP_VSEL_1_43_V** (2u)
- #define **PD_CMP_VSEL_0_655_V** (3u)
- #define **PD_CMP_VSEL_0_8_V** (4u)
- #define **PD_CMP_VSEL_1_235_V** (5u)
- #define **PD_CMP_VSEL_1_77_V** (6u)
- #define **PD_CMP_VSEL_2_6_V** (7u)
- #define **DN_COMP_IDX** (0u)
- #define **UP_COMP_IDX** (1u)
- #define **LOWER_LIMIT_IDX** (0u)
- #define **HIGHER_LIMIT_IDX** (1u)
- #define **RD_ROW_NO** (3u)
- #define **RD_COL_WIDTH** (4u)
- #define **RP_COL_WIDTH** (2u)
- #define **PDSS_TX_STOP_ON_SWAP_MASK** (1u << 26u)
- #define **FRS_TX_IRQ_WIDTH** (450u)
- #define **FRS_TX_SWAP_CTRL1_DFLT_VAL**
- #define **PDSS_HPD_CTRL1_DEFAULT_VALUE** (0x80530096)
- #define **PDSS_HPD_CTRL3_DEFAULT_VALUE** (0x0005304B)
- #define **PD_ADC_DEFAULT_VDDD_VOLT_MV** (5000u)
- #define **PD_ADC_MAX_VALUE** (0xFFu)
- #define **PD_ADC_NUM_LEVELS** (PD_ADC_MAX_VALUE + 1u)
- #define **PD_ADC_BAND_GAP_VOLT_MV** (1200u)
- #define **PD_ADC_LEVEL_MIN_THRESHOLD** (4u)
- #define **PD_ADC_LEVEL_MAX_THRESHOLD** (254u)
- #define **PD_ADC_TIMEOUT_COUNT** (200u)
- #define **MX_PD_ADC_REF_VOLT_MV** (2000u)
- #define **VBUS_OCP_MODE_EXT** (0u)
- #define **VBUS_OCP_MODE_INT** (1u)
- #define **VBUS_OCP_MODE_INT_AUTOCTRL** (2u)
- #define **VBUS_OCP_MODE_INT_SW_DB** (3u)
- #define **VBUS_OCP_MODE_POLLING** (4u)
- #define **VBUS_OCP_GPIO_ACTIVE_LOW** (0u)
- #define **VBUS_OCP_GPIO_ACTIVE_HIGH** (1u)

Typedefs

- typedef void(* **PD_ADC_CB_T**) (uint8_t port, bool comp_out)
- typedef void(* **pd_phy_cbk_t**) (uint8_t port, **pd_phy_evt_t** event)
- typedef void(* **vbus_cf_cbk_t**) (uint8_t port, bool cf_state)
- typedef void(* **pd_cmp_cbk_t**) (uint8_t port, bool state)
- typedef void(* **pd_supply_change_cbk_t**) (uint8_t port, **ccg_supply_t** supply_id, bool present)

Enumerations

- enum `PD_ADC_ID_T` { `PD_ADC_ID_0`, `PD_ADC_ID_1`, `PD_ADC_ID_OVUV`, `PD_ADC_NUM_ADC` }
- enum `PD_ADC_INPUT_T` { `PD_ADC_INPUT_AMUX_A`, `PD_ADC_INPUT_AMUX_B`, `PD_ADC_INPUT_BANDGAP`, `PD_ADC_INPUT_BJT`, `PD_ADC_NUM_INPUT` }
- enum `PD_ADC_INT_T` { `PD_ADC_INT_DISABLED`, `PD_ADC_INT_FALLING`, `PD_ADC_INT_RISING`, `PD_ADC_INT_BOTH` }
- enum `pd_phy_evt_t` { `PD_PHY_EVT_TX_MSG_COLLISION`, `PD_PHY_EVT_TX_MSG_PHY_IDLE`, `PD_PHY_EVT_TX_MSG_FAILED`, `PD_PHY_EVT_TX_MSG_SUCCESS`, `PD_PHY_EVT_RX_RST_COLLISION`, `PD_PHY_EVT_RX_RST_SUCCESS`, `PD_PHY_EVT_RX_MSG`, `PD_PHY_EVT_RX_MSG_CMPLT`, `PD_PHY_EVT_RX_RST`, `PD_PHY_EVT_FRS_SIG_RCVD`, `PD_PHY_EVT_FRS_SIG_SENT` }
- enum `vbus_ovp_mode_t` { `VBUS_OVP_MODE_ADC`, `VBUS_OVP_MODE_UVOV`, `VBUS_OVP_MODE_UVOV_AUTOCTRL` }
- enum `vbus_uvp_mode_t` { `VBUS_UVP_MODE_ADC`, `VBUS_UVP_MODE_INT_COMP`, `VBUS_UVP_MODE_INT_COMP_AUTOCTRL` }
- enum `frs_tx_source_t` { `FRS_TX_SOURCE_CPU` = 0, `FRS_TX_SOURCE_GPIO`, `FRS_TX_SOURCE_A↔DC1`, `FRS_TX_SOURCE_ADC2` }
- enum `frs_vsafe5v_source_t` { `FRS_RX_SOURCE_ADC1` = 0, `FRS_RX_SOURCE_ADC2` }
- enum `pd_fet_dr_t` { `PD_FET_DR_ACTIVE_HIGH` = 0, `PD_FET_DR_ACTIVE_LOW`, `PD_FET_DR_N_JN↔_FET`, `PD_FET_DR_P_JN_FET` }
- enum `sbu_switch_state_t` { `SBU_NOT_CONNECTED`, `SBU_CONNECT_AUX1`, `SBU_CONNECT_AUX2`, `SBU_CONNECT_LSTX`, `SBU_CONNECT_LSRX`, `SBU_MAX_STATE` }
- enum `aux_resistor_config_t` { `AUX_NO_RESISTOR` = 0, `AUX_1_1MEG_PU_RESISTOR` = 7, `AUX_1_100K_PD_RESISTOR` = 8, `AUX_1_470K_PD_RESISTOR` = 9, `AUX_2_100K_PU_RESISTOR` = 10, `AUX_2_4P7MEG_PD_RESISTOR` = 11, `AUX_2_1MEG_PD_RESISTOR` = 12, `AUX_MAX_RESISTOR_CONFIG` }
- enum `lscsa_app_config_t` { `LSCSA_OCP_CONFIG`, `LSCSA_EA_CONFIG`, `LSCSA_PFC_OFF_CONFIG`, `LSCSA_PFC_ON_CONFIG`, `LSCSA_SR_OFF_CONFIG`, `LSCSA_SR_ON_CONFIG`, `LSCSA_MAX_CONFIG_VALUE` }
- enum `comp_id_t` { `COMP_ID_UV` = 0, `COMP_ID_OV` = 1, `COMP_ID_VBUS_C_MON` = 2, `COMP_ID_VBUS_MON` = 2, `COMP_ID_VBUS_DISCHARGE` = 3, `COMP_ID_VSYS_DET` = 3, `COMP_ID_LSCSA_SCP` = 4, `COMP_ID_LSCSA_OCP` = 5, `COMP_ID_LSCSA_PFC` = 6, `COMP_ID_VSRC_NEW_P` = 7, `COMP_ID_VSRC_NEW_M` = 8, `COMP_ID_MAX` = 9 }
- enum `comp_tr_id_t` { `COMP_TR_ID_SR` = 0, `COMP_TR_ID_MAX` = 9 }
- enum `filter_id_t` { `FILTER_ID_UV` = 0, `FILTER_ID_OV` = 1, `FILTER_ID_DISCH_EN` = 2, `FILTER_ID_VBUS_MON` = 2, `FILTER_ID_LSCSA_SCP` = 3, `FILTER_ID_LSCSA_OCP` = 4, `FILTER_ID_LSCSA_PFC` = 5, `FILTER_ID_LSCSA_SR` = 6, `FILTER_ID_MAX` = 7 }
- enum `filter_edge_detect_cfg_t` { `FILTER_CFG_POS_DIS_NEG_DIS`, `FILTER_CFG_POS_DIS_NEG_EN`, `FILTER_CFG_POS_EN_NEG_DIS`, `FILTER_CFG_POS_EN_NEG_EN`, `FILTER_CFG_MAX` }
- enum `dpdm_mux_cfg_t` { `DPDM_MUX_CONN_NONE` = 0x00, `DPDM_MUX_CONN_USB_TOP` = 0x11, `DPDM_MUX_CONN_UART_TOP` = 0x22, `DPDM_MUX_CONN_USB_BOT` = 0x44, `DPDM_MUX_CONN_UART_BOT` = 0x88, `DPDM_MUX_CONN_USB_TOP_UART` = 0x99, `DPDM_MUX_CONN_USB_BOT_UART` = 0x66 }
- enum `ccg_supply_t` { `CCG_SUPPLY_VSYS` = 0x00, `CCG_SUPPLY_V5V` = 0x01 }

Functions

- `ccg_status_t pd_hal_init (uint8_t port)`
- `bool pd_phy_deepsleep (uint8_t port)`
- `bool pd_phy_wakeup (void)`
- `void pd_hal_cleanup (uint8_t port)`
- `void pd_hal_set_supply_change_evt_cb (pd_supply_change_cbk_t cb)`
- `ccg_status_t pd_phy_init (uint8_t port, pd_phy_cbk_t cbk)`
- `void pd_phy_refresh_roles (uint8_t port)`
- `void pd_phy_en_unchunked_tx (uint8_t port)`
- `void pd_phy_dis_unchunked_tx (uint8_t port)`
- `bool pd_phy_load_msg (uint8_t port, sop_t sop, uint8_t retries, uint8_t dobj_count, uint32_t header, bool unchunked, uint32_t *buf)`
- `bool pd_phy_send_msg (uint8_t port)`
- `void pd_phy_reset_rx_tx_sm (uint8_t port)`
- `ccg_status_t pd_phy_send_bist_cm2 (uint8_t port)`
- `ccg_status_t pd_phy_abort_bist_cm2 (uint8_t port)`
- `ccg_status_t pd_phy_abort_tx_msg (uint8_t port)`
- `ccg_status_t pd_phy_send_reset (uint8_t port, sop_t sop)`
- `pd_packet_extd_t * pd_phy_get_rx_packet (uint8_t port)`
- `bool pd_phy_is_busy (uint8_t port)`
- `ccg_status_t pd_typec_init (uint8_t port)`
- `ccg_status_t pd_typec_start (uint8_t port)`
- `ccg_status_t pd_typec_stop (uint8_t port)`
- `void pd_typec_en_rp (uint8_t port, uint8_t channel, rp_term_t rp_val)`
- `void pd_typec_dis_rp (uint8_t port, uint8_t channel)`
- `void pd_typec_en_dpslp_rp (uint8_t port)`
- `void pd_typec_dis_dpslp_rp (uint8_t port)`
- `void pd_typec_en_rd (uint8_t port, uint8_t channel)`
- `void pd_typec_rd_enable (uint8_t port)`
- `void pd_typec_dis_rd (uint8_t port, uint8_t channel)`
- `void pd_typec_en_deadbat_rd (uint8_t port)`
- `void pd_typec_snk_update_trim (uint8_t port)`
- `cc_state_t pd_typec_get_cc_status (uint8_t port)`
- `void pd_typec_set_polarity (uint8_t port, bool polarity)`
- `ccg_status_t pd_vconn_enable (uint8_t port, uint8_t channel)`
- `ccg_status_t pd_vconn_disable (uint8_t port, uint8_t channel)`
- `void pd_hal_vconn_ocp_enable (uint8_t port, uint8_t debounce, PD_ADC_CB_T cb)`
- `void pd_hal_vconn_ocp_disable (uint8_t port)`
- `bool pd_is_vconn_present (uint8_t port, uint8_t channel)`
- `uint8_t pd_adc_volt_to_level (uint8_t port, PD_ADC_ID_T adc_id, uint16_t volt)`
- `uint16_t pd_adc_level_to_volt (uint8_t port, PD_ADC_ID_T adc_id, uint8_t level)`
- `uint16_t pd_adc_get_vbus_voltage (uint8_t port, PD_ADC_ID_T adc_id, uint8_t level)`
- `ccg_status_t pd_adc_free_run_ctrl (uint8_t port, PD_ADC_ID_T adc_id, PD_ADC_INPUT_T input, uint8_t level)`
- `void pd_adc_comparator_ctrl (uint8_t port, PD_ADC_ID_T adc_id, PD_ADC_INPUT_T input, uint8_t level, PD_ADC_INT_T int_cfg, PD_ADC_CB_T cb)`
- `bool pd_adc_comparator_sample (uint8_t port, PD_ADC_ID_T adc_id, PD_ADC_INPUT_T input, uint8_t level)`
- `bool pd_adc_get_comparator_status (uint8_t port, PD_ADC_ID_T adc_id)`
- `uint8_t pd_adc_sample (uint8_t port, PD_ADC_ID_T adc_id, PD_ADC_INPUT_T input)`
- `uint16_t pd_adc_calibrate (uint8_t port, PD_ADC_ID_T adc_id)`
- `ccg_status_t pd_adc_init (uint8_t port, PD_ADC_ID_T adc_id)`
- `uint8_t pd_get_vbus_adc_level (uint8_t port, PD_ADC_ID_T adc_id, uint16_t volt, int8_t per)`
- `void pd_hal_config_auto_toggle (uint8_t port, bool enable)`

- bool `pd_hal_is_auto_toggle_active` (uint8_t port)
- void `pd_hal_abort_auto_toggle` (uint8_t port)
- void `pd_hal_typec_sm_restart` (uint8_t port)
- uint16_t `pd_hal_measure_vbus` (uint8_t port)
- void `pd_hal_set_fet_drive` (`pd_fet_dr_t` pctrl_drive, `pd_fet_dr_t` cctrl_drive)
- void `pd_hal_dual_fet_config` (bool dual_fet, uint8_t spacing)
- void `pd_hal_set_cc_ovp_pending` (uint8_t port)
- void `pd_hal_set_vbus_detach_params` (`PD_ADC_ID_T` adc_id, `PD_ADC_INPUT_T` adc_inp)
- `PD_ADC_ID_T` `pd_hal_get_vbus_detach_adc` (void)
- `PD_ADC_INPUT_T` `pd_hal_get_vbus_detach_input` (void)
- void `pd_hal_set_vbus_mon_divider` (uint8_t divider)
- void `pd_lscka_calc_cfg` (uint32_t cur_10mA, uint8_t *gain_sel, uint8_t *vref_sel)
- `ccg_status_t` `pd_lscka_cfg` (`lscka_app_config_t` lscka_app, uint8_t gain_sel)
- void `pd_cf_enable` (uint8_t port, uint32_t cur, `vbus_cf_cbk_t` cbk)
- void `pd_cf_disable` (uint8_t port)
- bool `pd_cf_get_status` (uint8_t port)
- void `solt_no_vsys_handler` (void)
- void `solt_vsys_removal_handler` (void)
- bool `pd_set_sr_comp` (uint8_t port, uint32_t cur, `filter_edge_detect_cfg_t` edge, `pd_cmp_cbk_t` cbk)
- void `pd_stop_sr_comp` (uint8_t port)
- bool `pd_set_pfc_comp` (uint8_t port, uint32_t cur, `filter_edge_detect_cfg_t` edge, `pd_cmp_cbk_t` cbk)
- void `pd_stop_pfc_comp` (uint8_t port)
- bool `pd_cmp_get_status` (uint8_t port, `filter_id_t` id, bool is_filtered)

6.26.1 Detailed Description

CCG PD PHY driver module header file.

6.26.2 Macro Definition Documentation

6.26.2.1 FRS_TX_SWAP_CTRL1_DFLT_VAL

```
#define FRS_TX_SWAP_CTRL1_DFLT_VAL
```

Value:

```
(FRS_TX_IRQ_WIDTH | PDSS_SWAPT_CTRL1_DEFAULT_LEVEL | \
(2u << PDSS_SWAPT_CTRL1_COMMAND_POS) | \
PDSS_SWAPT_CTRL1_RESET_SWAPT_STATE)
```

6.26.2.2 HEADER_INFO_CFG

```
#define HEADER_INFO_CFG
```

Value:

```
((EXTENDED_DATA_BIT_LOC_VAL << 8) | \
(EXTENDED_DATA_BYTE_FIRST_BIT_LOCATION_VAL << 12) | \
(EXTENDED_DATA_BYTE_LAST_BIT_LOCATION_VAL << 17) | \
(CHUNK_BIT_LOCATION_VAL << 22))
```

6.26.2.3 RCV_INTR_MASK

```
#define RCV_INTR_MASK
```

Value:

```
(PDSS_INTR0_RX_STATE_IDLE | \
    PDSS_INTR0_EOP_ERROR | \
    PDSS_INTR0_RCV_GOOD_PACKET_COMPLETE | \
    PDSS_INTR0_RCV_BAD_PACKET_COMPLETE | \
    PDSS_INTR0_RCV_GOODCRC_MSG_COMPLETE | \
    PDSS_INTR0_RCV_EXPT_GOODCRC_MSG_COMPLETE | \
    PDSS_INTR0_RX_SOP | \
    PDSS_INTR0_RX_SRAM_HALF_END | \
    PDSS_INTR0_RX_OVER_RUN)
```

6.26.2.4 RST_TX_INTERRUPTS

```
#define RST_TX_INTERRUPTS
```

Value:

```
(PDSS_INTR0_TX_HARD_RST_DONE | \
    PDSS_INTR0_COLLISION_TYPE4)
```

Reception detected at reset transmit.

6.26.2.5 RX_CC_CFG

```
#define RX_CC_CFG
```

Value:

```
(RX_CNT_MAX_CFG | \
    RX_UI_BOUNDARY_DELTA_CFG)
```

6.26.2.6 RX_INTERRUPTS

```
#define RX_INTERRUPTS
```

Value:

```
(PDSS_INTR0_TX_GOODCRC_MSG_DONE | \
    PDSS_INTR0_RX_STATE_IDLE | \
    PDSS_INTR0_RCV_RST | \
    PDSS_INTR0_COLLISION_TYPE3)
```

Reception detected at GoodCRC transmit.

6.26.2.7 RX_ORDER_SET_CTRL_CFG

```
#define RX_ORDER_SET_CTRL_CFG
```

Value:

```
(PDSS_RX_ORDER_SET_CTRL_SOP_CMP_OPT | \
    PDSS_RX_ORDER_SET_CTRL_RST_CMP_OPT | \
    PDSS_RX_ORDER_SET_CTRL_PREAMBLE_SOP_EN | \
    PDSS_RX_ORDER_SET_CTRL_PREAMBLE_RST_EN)
```

6.26.2.8 TX_INTERRUPTS

```
#define TX_INTERRUPTS
```

Value:

```
(PDSS_INTR0_RCV_GOODCRC_MSG_COMPLETE | \
    PDSS_INTR0_CRC_RX_TIMER_EXP | \
    PDSS_INTR0_COLLISION_TYPE1 | \
    PDSS_INTR0_TX_PACKET_DONE | \
    PDSS_INTR0_COLLISION_TYPE2)
```

Reception detected at message retransmit.

6.26.2.9 VBUS_OCP_GPIO_ACTIVE_HIGH

```
#define VBUS_OCP_GPIO_ACTIVE_HIGH (1u)
```

GPIO high indicates fault condition.

6.26.2.10 VBUS_OCP_GPIO_ACTIVE_LOW

```
#define VBUS_OCP_GPIO_ACTIVE_LOW (0u)
```

GPIO low indicates fault condition.

6.26.2.11 VBUS_OCP_MODE_EXT

```
#define VBUS_OCP_MODE_EXT (0u)
```

OCP through external hardware.

6.26.2.12 VBUS_OCP_MODE_INT

```
#define VBUS_OCP_MODE_INT (1u)
```

Internal OCP without software debounce or hardware gate control.

6.26.2.13 VBUS_OCP_MODE_INT_AUTOCTRL

```
#define VBUS_OCP_MODE_INT_AUTOCTRL (2u)
```

Internal OCP with hardware gate control.

6.26.2.14 VBUS_OCP_MODE_INT_SW_DB

```
#define VBUS_OCP_MODE_INT_SW_DB (3u)
```

Internal OCP with software debounce.

6.26.2.15 VBUS_OCP_MODE_POLLING

```
#define VBUS_OCP_MODE_POLLING (4u)
```

Solution level polling based OCP.

6.26.3 Typedef Documentation

6.26.3.1 PD_ADC_CB_T

`PD_ADC_CB_T`

PD ADC comparator interrupt callback type. This callback is called when the desired ADC event/interrupt occurs.
Available events:

- true : Input voltage is higher than the reference.
- false : Input voltage is lower than the reference.

Parameters

<i>port</i>	PD port on which the ADC event occurred.
<i>comp_out</i>	Specifies the type of event.

6.26.3.2 pd_cmp_cbk_t

`pd_cmp_cbk_t`

Comparator interrupt callback function.

Parameters

<i>port</i>	PD port on which the event occurred.
<i>state</i>	State of the comparator.

6.26.3.3 pd_phy_cbk_t

`pd_phy_cbk_t`

PD PHY callback prototype. This function will be used to notify the stack about PHY events.

Parameters

<i>port</i>	PD port on which the PHY event occurred.
<i>event</i>	Type of PD PHY event.

6.26.3.4 pd_supply_change_cbk_t

`pd_supply_change_cbk_t`

Callback function used to provide notification about input supply changes.

Parameters

<i>port</i>	PD port associated with the supply.
<i>supply_id</i>	ID of the supply on which change is detected.
<i>present</i>	Whether the identified supply is now present (true) or absent (false).

6.26.3.5 vbus_cf_cbk_t

```
vbus_cf_cbk_t
```

VBus current foldback callback function.

Parameters

<i>port</i>	PD port on which the event occurred.
<i>cf_state</i>	Whether in CF mode or not.

6.26.4 Enumeration Type Documentation**6.26.4.1 aux_resistor_config_t**

```
enum aux_resistor_config_t
```

Enum to hold resistor configuration for AUX1 and AUX2. Values assigned are the bit position of corresponding configuration in sbu_ctrl register.

Enumerator

AUX_NO_RESISTOR	No resistor.
AUX_1_1MEG_PU_RESISTOR	AUX1 1M0hm Pullup resistor.
AUX_1_100K_PD_RESISTOR	AUX1 100KOhm Pulldown resistor.
AUX_1_470K_PD_RESISTOR	AUX1 470KOhm Pulldown resistor.
AUX_2_100K_PU_RESISTOR	AUX2 100KOhm Pullup resistor.
AUX_2_4P7MEG_PD_RESISTOR	AUX2 4.7M0hm Pulldown resistor.
AUX_2_1MEG_PD_RESISTOR	AUX2 1M0hm Pulldown resistor.
AUX_MAX_RESISTOR_CONFIG	Not supported.

6.26.4.2 ccg_supply_t

```
enum ccg_supply_t
```

List of power supplies input to and monitored by the CCGx device.

Enumerator

CCG_SUPPLY_VSYS	Vsys supply. Applies to parts that can be powered off VBus also (CCG3, CCG5, CCG3PA)
CCG_SUPPLY_V5V	V5V input used to derive the VConn supply from. Can be port-specific.

6.26.4.3 comp_id_t

```
enum comp_id_t
```

IDs of comparators in CCG3PA, CCG3PA2, CCG5.

Enumerator

COMP_ID_UV	UV comparator. Common for CCG3PA/CCG5.
COMP_ID_OV	OV comparator. Common for CCG3PA/CCG5.
COMP_ID_VBUS_C_MON	VBUS_C_MON comparator. CCG3PA only.
COMP_ID_VBUS_MON	VBUS_MON comparator. CCG5 only.
COMP_ID_VBUS_DISCHARGE	Discharge comparator. CCG3PA only.
COMP_ID_VSYS_DET	Vsys detection. CCG5 only.
COMP_ID_LSCSA_SCP	SCP comparator. CCG3PA only.
COMP_ID_LSCSA_OCP	OCP comparator. CCG3PA only.
COMP_ID_LSCSA_PFC	PFC comparator. CCG3PA only.
COMP_ID_VSRC_NEW_P	VSRC_NEW comparator. CCG3PA only.
COMP_ID_VSRC_NEW_M	VSRC_NEW comparator. CCG3PA only.
COMP_ID_MAX	End of comparator list.

6.26.4.4 comp_tr_id_t

```
enum comp_tr_id_t
```

IDs of trim enabled comparators in CCG3PA, CCG3PA2, CCG5.

Enumerator

COMP_TR_ID_SR	SR comparator. CCG3PA only.
COMP_TR_ID_MAX	End of comparator list.

6.26.4.5 dpdm_mux_cfg_t

```
enum dpdm_mux_cfg_t
```

List of possible settings for the DP/DM MUX on the CCG5 device.

Enumerator

DPDM_MUX_CONN_NONE	No connections enabled through the DPDM Mux.
--------------------	--

Enumerator

DPDM_MUX_CONN_USB_TOP	Connect D+/D- to D+/D- on the top side of the connector.
DPDM_MUX_CONN_UART_TOP	Connect UART TX/RX to D+/D- on the top side of the connector.
DPDM_MUX_CONN_USB_BOT	Connect D+/D- to D+/D- on the bottom side of the connector.
DPDM_MUX_CONN_UART_BOT	Connect UART TX/RX to D+/D- on the bottom side of the connector.
DPDM_MUX_CONN_USB_TOP_UART	Connect D+/D- to top and UART TX/RX to bottom side.
DPDM_MUX_CONN_USB_BOT_UART	Connect D+/D- to bottom and UART TX/RX to top side.

6.26.4.6 filter_edge_detect_cfg_t

```
enum filter_edge_detect_cfg_t
```

Enumerator

FILTER_CFG_POS_DIS_NEG_DIS	Interrupt disabled.
FILTER_CFG_POS_DIS_NEG_EN	Negative edge detection interrupt.
FILTER_CFG_POS_EN_NEG_DIS	Positive edge detection interrupt.
FILTER_CFG_POS_EN_NEG_EN	Both edge detection interrupt.

6.26.4.7 filter_id_t

```
enum filter_id_t
```

IDs of Filters in CCG3PA.

Filter edge detect configuration list.

Enumerator

FILTER_ID_UV	UV filter, common for CCG3PA and CCG5.
FILTER_ID_OV	OV filter, common for CCG3PA and CCG5.
FILTER_ID_DISCH_EN	Discharge enable filter for CCG3PA.
FILTER_ID_VBUS_MON	VBus monitor filter for CCG5.
FILTER_ID_LSCSA_SCP	SCP filter for CCG3PA.
FILTER_ID_LSCSA_OCP	OCP filter for CCG3PA.
FILTER_ID_LSCSA_PFC	PFC filter for CCG3PA.
FILTER_ID_LSCSA_SR	SR filter for CCG3PA.
FILTER_ID_MAX	Number of supported filters.

6.26.4.8 lscsa_app_config_t

```
enum lscsa_app_config_t
```

Enumeration of all possible Low Side CSA applications.

Enumerator

LSCSA_OCP_CONFIG	OCP comparator gain control CCG3PA only.
LSCSA_EA_CONFIG	EA comparator gain control CCG3PA only.
LSCSA_PFC_OFF_CONFIG	PFC OFF comparator gain control CCG3PA only.
LSCSA_PFC_ON_CONFIG	PFC ON comparator gain control CCG3PA only.
LSCSA_SR_OFF_CONFIG	SR OFF comparator gain control CCG3PA only.
LSCSA_SR_ON_CONFIG	SR ON comparator gain control CCG3PA only.
LSCSA_MAX_CONFIG_VALUE	End of comparator list.

6.26.4.9 PD_ADC_ID_T`enum PD_ADC_ID_T`

ADC block IDs on the CCG device.

Enumerator

PD_ADC_ID_0	ADC-0 associated with the PD block.
PD_ADC_ID_1	ADC-1 associated with the PD block.
PD_ADC_ID_OVUV	Dedicated ADC for Over-Voltage/Under-Voltage detection. Only available in the CCG3 device.
PD_ADC_NUM_ADC	Maximum number of ADCs in the PD block.

6.26.4.10 PD_ADC_INPUT_T`enum PD_ADC_INPUT_T`

Enumeration of PD ADC input sources. Refer to CCG device datasheet and TRM for more details.

Enumerator

PD_ADC_INPUT_AMUX_A	AMUX_A bus.
PD_ADC_INPUT_AMUX_B	AMUX_B bus.
PD_ADC_INPUT_BANDGAP	BANDGAP input.
PD_ADC_INPUT_BJT	BJT.
PD_ADC_NUM_INPUT	Number of ADC inputs available.

6.26.4.11 PD_ADC_INT_T`enum PD_ADC_INT_T`

PD comparator interrupt configuration enumeration. Note: These are the settings for INTR_1_CFG ADC output, not ADC_SAR_CTRL.

Enumerator

PD_ADC_INT_DISABLED	Comparator interrupt disabled.
---------------------	--------------------------------

Enumerator

PD_ADC_INT_FALLING	Comparator interrupt on falling edge.
PD_ADC_INT_RISING	Comparator interrupt on rising edge.
PD_ADC_INT_BOTH	Comparator interrupt on either edge.

6.26.4.12 pd_phy_evt_t

```
enum pd_phy_evt_t
```

PD PHY state enumeration.

Enumerator

PD_PHY_EVT_TX_MSG_COLLISION	Bus busy at message transmission.
PD_PHY_EVT_TX_MSG_PHY_IDLE	Bus idle, ready for message transmission.
PD_PHY_EVT_TX_MSG_FAILED	Message transmission was not successful.
PD_PHY_EVT_TX_MSG_SUCCESS	Message transmission was successful.
PD_PHY_EVT_TX_RST_COLLISION	Bus busy just before reset transmission.
PD_PHY_EVT_TX_RST_SUCCESS	Reset transmission was successful.
PD_PHY_EVT_RX_MSG	Message received.
PD_PHY_EVT_RX_MSG_CMPLT	Message received and GoodCRC sent aka collision type 3.
PD_PHY_EVT_RX_RST	Reset was received.
PD_PHY_EVT_FRS_SIG_RCVD	FRS signal was received.
PD_PHY_EVT_FRS_SIG_SENT	FRS signal was transmitted.

6.26.4.13 sbu_switch_state_t

```
enum sbu_switch_state_t
```

Enum to hold SBU connection state for CCG3/CCG5. CCG3 and CCG5 provide internal switch to route SBU1/SBU2 signals to AUX_P/AUX_N, LSTX/LSRX, or Isolate.

Enumerator

SBU_NOT_CONNECTED	SBU pin is isolated.
SBU_CONNECT_AUX1	Connect SBU pin to AUX_P.
SBU_CONNECT_AUX2	Connect SBU pin to AUX_N.
SBU_CONNECT_LSTX	Connect SBU pin to LSTX.
SBU_CONNECT_LSRX	Connect SBU pin to LSRX.
SBU_MAX_STATE	Invalid value: not supported.

6.26.4.14 vbus_ovp_mode_t

```
enum vbus_ovp_mode_t
```

CCG OVP modes enumeration.

Enumerator

VBUS_OVP_MODE_ADC	OVP using CCG internal ADC.
VBUS_OVP_MODE_UVOV	OVP using the UVOV block on CCGx. Actual gate control is done by firmware.
VBUS_OVP_MODE_UVOV_AUTOCTRL	OVP using the OVOV block with automatic gate driver control.

6.26.4.15 vbus_uvp_mode_t

enum `vbus_uvp_mode_t`

CCG UVP modes enumeration.

Enumerator

VBUS_UVP_MODE_ADC	UVP using CCG internal ADC.
VBUS_UVP_MODE_INT_COMP	UVP using internal comparartor
VBUS_UVP_MODE_INT_COMP_AUTOCTRL	UVP using internal comparator with automatic gate driver control.

6.26.5 Function Documentation

6.26.5.1 pd_adc_calibrate()

```
uint16_t pd_adc_calibrate (
    uint8_t port,
    PD_ADC_ID_T adc_id )
```

This function calibrates the specified ADC for operation.

This function calibrates the specified ADC by identifying the VDDD voltage for reference. It should be noted that by calling the function, the previously calculated threshold levels may have to be changed based on the VDDD reading. The VDDD level is calculated based on the bandgap voltage which is expected to be constant.

Parameters

<code>port</code>	Port index. Caller should ensure to provide only valid values.
<code>adc_id</code>	ADC ID.

Returns

Returns the VDDD value in mV after calibration.

6.26.5.2 pd_adc_comparator_ctrl()

```
void pd_adc_comparator_ctrl (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    PD_ADC_INPUT_T input,
    uint8_t level,
    PD_ADC_INT_T int_cfg,
    PD_ADC_CB_T cb )
```

This function configures the ADC for comparator functionality with the requested threshold.

This function configures the ADC block as a comparator. The function takes the input to be configured and the ADC comparator threshold. It also takes a callback. If the callback is not NULL, then the threshold is configured and interrupts are enabled. If the callback is NULL, then the ADC / comparator is set to the low power state and interrupts are disabled.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>input</i>	ADC input source.
<i>level</i>	Comparator level.
<i>int_cfg</i>	Interrupt configuration.
<i>cb</i>	Callback on interrupt.

Returns

None.

6.26.5.3 pd_adc_comparator_sample()

```
bool pd_adc_comparator_sample (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    PD_ADC_INPUT_T input,
    uint8_t level )
```

This function temporarily configures the comparator as requested and takes a sample.

This function restores the state of the comparator after operation. This is useful when the comparator is already configured to function with a certain input and level with interrupt and another reading needs to be done without having to re-configure the block after the sampling.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>input</i>	ADC input source.

Returns

Returns true if voltage > level, false otherwise.

6.26.5.4 pd_adc_free_run_ctrl()

```
ccg_status_t pd_adc_free_run_ctrl (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    PD_ADC_INPUT_T input,
    uint8_t level )
```

This function configures the ADC for comparator functionality with the requested threshold with no interrupts.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>input</i>	ADC input source.
<i>level</i>	Comparator level.

Returns

ccg_status_t

6.26.5.5 pd_adc_get_comparator_status()

```
bool pd_adc_get_comparator_status (
    uint8_t port,
    PD_ADC_ID_T adc_id )
```

This function gets the current comparator status.

This function does not configure the ADC / comparator. It just returns the current state of the comparator. If true is returned, then the input voltage is greater than the reference and if false, the input voltage is lower than the reference.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.

Returns

Returns the comparator output.

6.26.5.6 pd_adc_get_vbus_voltage()

```
uint16_t pd_adc_get_vbus_voltage (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    uint8_t level )
```

This function converts the ADC level to VBUS voltage in millivolts.

It takes an 8-bit ADC reading and returns the corresponding 16-bit voltage value in millivolts. This function does not perform any ADC operations. port and adc_id are not used for pdss_mx_hal only used for pdss_hal.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>level</i>	The 8-bit ADC reading.

Returns

Returns voltage in mV.

6.26.5.7 pd_adc_init()

```
ccg_status_t pd_adc_init (
    uint8_t port,
    PD_ADC_ID_T adc_id )
```

This function initializes the PD ADC block.

This function enables the PD block and the registers required for ADC operation. It then calibrates the ADC to identify the VDDD voltage. This function does not start any ADC operations.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.

Returns

ccg_status_t

6.26.5.8 pd_adc_level_to_volt()

```
uint16_t pd_adc_level_to_volt (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    uint8_t level )
```

This function converts the ADC units to voltage in millivolts.

It takes an 8-bit ADC reading and returns the corresponding 16-bit voltage value in millivolts. This function does not perform any ADC operations.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>level</i>	The 8-bit ADC reading.

Returns

Returns voltage in mV.

6.26.5.9 pd_adc_sample()

```
uint8_t pd_adc_sample (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    PD_ADC_INPUT_T input )
```

This function samples the ADC.

This function enables the ADC block to function as an ADC and returns the sample value in ADC units. This function disables any previously configured comparator interrupts / settings before sampling and restores them after the sampling is done. If any interrupt scenario happens across the sampling, the information is lost.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>input</i>	ADC input source.

Returns

Returns the ADC sample.

6.26.5.10 pd_adc_volt_to_level()

```
uint8_t pd_adc_volt_to_level (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    uint16_t volt )
```

This function converts the voltage in millivolt to ADC units.

It takes a 16-bit voltage value in millivolts and returns the corresponding 8-bit ADC reading. This function does not perform any ADC operations.

The minimum value is limited by the PD_ADC_LEVEL_MIN_THRESHOLD value.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_↔_{_id}</i>	ADC ID.
<i>volt</i>	Voltage in mV.

Returns

Returns the 8-bit ADC reading.

6.26.5.11 pd_cf_disable()

```
void pd_cf_disable (
    uint8_t port )
```

Disables VBus Current foldback on the specified port.

Parameters

<i>port</i>	USB-PD port.
-------------	--------------

Returns

None

6.26.5.12 pd_cf_enable()

```
void pd_cf_enable (
    uint8_t port,
    uint32_t cur,
    vbus_cf_cbk_t cbk )
```

Enables VBus Current foldback on the specified port.

Parameters

<i>port</i>	USB-PD port on which to enable CF. Operating current in 10mA units.
<i>cbk</i>	Function to be called when CF event is detected.

Returns

true if param correct else return false

6.26.5.13 pd_cf_get_status()

```
bool pd_cf_get_status (
    uint8_t port )
```

Retrieves the status of the CF mode hardware.

Parameters

<i>port</i>	USB-PD port.
-------------	--------------

Returns

Whether the hardware module is in CF mode or not.

6.26.5.14 pd_cmp_get_status()

```
bool pd_cmp_get_status (
    uint8_t port,
    filter_id_t id,
    bool is_filtered )
```

Gets the current status of the VBUS / LSCSA comparators.

Parameters

<i>port</i>	USB-PD port.
<i>id</i>	comparator filter ID for which the status has to be retrieved.
<i>is_filtered</i>	Whether to get the filtered or unfiltered status.

Returns

None

6.26.5.15 pd_get_vbus_adc_level()

```
uint8_t pd_get_vbus_adc_level (
    uint8_t port,
    PD_ADC_ID_T adc_id,
    uint16_t volt,
    int8_t per )
```

This function gets the ADC level that corresponds to the actual voltage on vbus. It also takes into account the VBus monitor divider.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>adc_id</i>	ADC ID.
<i>volt</i>	Voltage in 50mV units.
<i>per</i>	Percentage margin on the voltage.

Returns

Returns the ADC level.

6.26.5.16 pd_hal_abort_auto_toggle()

```
void pd_hal_abort_auto_toggle (
    uint8_t port )
```

Abort any ongoing automatic DRP toggle operation.

Parameters

<i>port</i>	Port on which toggle is to be aborted.
-------------	--

Returns

None

6.26.5.17 pd_hal_cleanup()

```
void pd_hal_cleanup (
    uint8_t port )
```

This function cleans up the PD block after a disconnect.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.18 pd_hal_config_auto_toggle()

```
void pd_hal_config_auto_toggle (
    uint8_t port,
    bool enable )
```

Enable/disable automatic hardware toggle operation as part of deep sleep cycle.

Parameters

<i>port</i>	PD port to be updated.
<i>enable</i>	Whether automatic toggle is to be enabled.

Returns

None

6.26.5.19 pd_hal_dual_fet_config()

```
void pd_hal_dual_fet_config (
    bool dual_fet,
    uint8_t spacing )
```

Configures FET parameters for CCG3 devices.

This function configures various FET control parameters for the device. It should be called before the PD HAL is initialized. Also it should be called only once during initialization. Since this is high voltage FET configuration, any wrong configuration shall result in damage of the device and boards. The function should be called only when the default behaviour needs to be overridden.

Caller should ensure that the FET access is not done before it is configured and is never configured while the PD stack is in operation.

CCG3 devices shall use dual FET configuration with spacing of 10LF cycles by default. If this function is not invoked, then the default configuration shall be used. CCG3 shall always function in dual FET mode as the FET controls are dedicated IOs.

Parameters

<i>dual_fet</i>	Set to true if the system uses dual FETs for each direction. Otherwise set to false.
<i>spacing</i>	Spacing in LF-cycles between dual FETs for firmware based turn-on and turn-off. Auto shut-off and turn-on happens simultaneously. Valid only for dual_fet configuration. Set to zero for simultaneous control.

Returns

None.

Warning

Misconfiguration of this parameter shall result in board damage.

6.26.5.20 pd_hal_get_vbus_detach_adc()

```
PD_ADC_ID_T pd_hal_get_vbus_detach_adc (
    void )
```

Identify the CCGx ADC that is used for VBus detach detection.

Returns

ID of the ADC block used for VBus detach detection.

6.26.5.21 pd_hal_get_vbus_detach_input()

```
PD_ADC_INPUT_T pd_hal_get_vbus_detach_input (
    void )
```

Identify the ADC input that is used for VBus detach detection.

Returns

ADC block input used for VBus detach detection.

6.26.5.22 pd_hal_init()

```
ccg_status_t pd_hal_init (
    uint8_t port )
```

This function initializes the PDSS IP with necessary clock and interrupt handlers.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

ccg_status_t.

6.26.5.23 pd_hal_is_auto_toggle_active()

```
bool pd_hal_is_auto_toggle_active (
    uint8_t port )
```

Check whether automatic DRP toggle operation is active.

Parameters

<i>port</i>	PD port to be checked.
-------------	------------------------

Returns

Current status of automatic DRP toggle operation.

6.26.5.24 pd_hal_measure_vbus()

```
uint16_t pd_hal_measure_vbus (
    uint8_t port )
```

Measure the current voltage on the VBus supply.

Parameters

<i>port</i>	PD port to be measured.
-------------	-------------------------

Returns

VBus voltage in mV units.

6.26.5.25 pd_hal_set_fet_drive()

```
void pd_hal_set_fet_drive (
    pd_fet_dr_t pctrl_drive,
    pd_fet_dr_t cctrl_drive )
```

Configures the drive modes for the FETs.

This function allows the application to select polarity of drive for CCG4 devices and N-FET drive or P-FET drive for CCG3 devices. The configuration should match the hardware implementation on the board.

CCG3 devices support N-FET by default for both PCTRL and CCTRL. Override this only for the boards with P-FETs. Standard Cypress reference schematics and kits use N-FETs and changing this shall result in board damage.

CCG4 devices support active high polarity for PCTRL (source) and active low polarity for CCTRL (sink) by default. This configuration matches the standard Cypress reference schematics and kits and changing this shall result in board damage.

This function is expected to be called once, before the pdss_hal is initialized. Calls during stack operation are not restricted but are unsupported and likely to result in spurious behaviour and / or board damage.

Parameters

<i>is_p_jn_fet</i>	Set to true if the system uses P-FETs, false otherwise.
--------------------	---

Returns

None.

Warning

Misconfiguration of this parameter will result in board damage.

6.26.5.26 pd_hal_set_supply_change_evt_cb()

```
void pd_hal_set_supply_change_evt_cb (
    pd_supply_change_cbk_t cb )
```

Register a callback that can be used for notification of power supply changes.

Parameters

<i>cb</i>	Callback function pointer.
-----------	----------------------------

Returns

None

6.26.5.27 pd_hal_set_vbus_detach_params()

```
void pd_hal_set_vbus_detach_params (
    PD_ADC_ID_T adc_id,
    PD_ADC_INPUT_T adc_inp )
```

Select the comparator block and input setting used for VBus detach detection. CCG will use the selected settings to detect Type-C disconnection based on removal of the VBus power.

Parameters

<i>adc_id</i>	Select the comparator (ADC) block to be used.
<i>adc_inp</i>	Select the comparator input to be used.

Returns

None

6.26.5.28 pd_hal_set_vbus_mon_divider()

```
void pd_hal_set_vbus_mon_divider (
    uint8_t divider )
```

Specify the voltage division ratio between the voltage applied on VBUS_MON and the actual VBus voltage. The commonly used resistor divider ratio used is 1:10, giving a voltage division ratio of 1/11.

Parameters

<i>divider</i>	Ratio between VBUS_MON voltage and VBus voltage.
----------------	--

Returns

None

6.26.5.29 pd_hal_typec_sm_restart()

```
void pd_hal_typec_sm_restart (
    uint8_t port )
```

Restart Type-C state machine once auto toggle operation is complete. This is normally required when the port has just exited the auto DRP toggle stage, and the state machine needs to handle further operations.

Parameters

<i>port</i>	Port on which state machine restart is required.
-------------	--

Returns

None

6.26.5.30 pd_hal_vconn_ocp_disable()

```
void pd_hal_vconn_ocp_disable (
    uint8_t port )
```

Disable Over-Current detection on the VConn power source.

Parameters

<i>port</i>	PD port to be configured.
-------------	---------------------------

Returns

None.

6.26.5.31 pd_hal_vconn_ocp_enable()

```
void pd_hal_vconn_ocp_enable (
    uint8_t port,
    uint8_t debounce,
    PD_ADC_CB_T cb )
```

Enable Over-Current detection on the VConn power source.

Parameters

<i>port</i>	PD port to be configured.
<i>debounce</i>	Debounce period in milliseconds.

Returns

None.

6.26.5.32 pd_is_vconn_present()

```
bool pd_is_vconn_present (
    uint8_t port,
    uint8_t channel )
```

This function gets Vconn status for the specified channel.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

Returns true if Vconn is turned on, false otherwise.

6.26.5.33 pd_lscsa_calc_cfg()

```
void pd_lscsa_calc_cfg (
    uint32_t cur_10mA,
    uint8_t * gain_sel,
    uint8_t * vref_sel )
```

Calculates the gain and reference settings for a specific current requirement. This function eliminates the precalculated table.

Parameters

<i>cur_10mA</i>	Current setting in 10mA unit for which CSA setting is required.
<i>vref_sel</i>	Pointer to variable where the voltage reference code corresponding to the current setting needs to be stored.
<i>av_sel</i>	Pointer to variable where the gain setting code corresponding to the current setting needs to be stored.

6.26.5.34 pd_lscsa_cfg()

```
ccg_status_t pd_lscsa_cfg (
    lscsa_app_config_t lscsa_app,
    uint8_t gain_sel )
```

Sets up LSCSA for specified application.

Parameters

<i>lscsa_app</i>	LSCSA application
<i>gain_sel_value</i>	Gain selection code to be used

Returns

ccg_status_t

6.26.5.35 pd_phy_abort_bist_cm2()

```
ccg_status_t pd_phy_abort_bist_cm2 (
    uint8_t port )
```

This function stops transmission of BIST Carrier Mode 2.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

ccg_status_t

6.26.5.36 pd_phy_abort_tx_msg()

```
ccg_status_t pd_phy_abort_tx_msg (
    uint8_t port )
```

This function stops transmission of a message.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

ccg_status_t

6.26.5.37 pd_phy_deepsleep()

```
bool pd_phy_deepsleep (
    uint8_t port )
```

This function configures the PD block for deepsleep entry.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

Returns true if the port is not busy and has been configured to go to deepsleep, false otherwise. Also returns true if the block was not enabled.

6.26.5.38 pd_phy_dis_unchunked_tx()

```
void pd_phy_dis_unchunked_tx (
    uint8_t port )
```

This function disables transmission of unchunked extended messages.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.39 pd_phy_en_unchunked_tx()

```
void pd_phy_en_unchunked_tx (
```

```
    uint8_t port )
```

This function enables transmission of unchunked extended messages.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.40 pd_phy_get_rx_packet()

```
pd_packet_extd_t* pd_phy_get_rx_packet (
    uint8_t port )
```

This function returns the received packet.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

Pointer to the received PD packet.

6.26.5.41 pd_phy_init()

```
ccg_status_t pd_phy_init (
    uint8_t port,
    pd_phy_cbk_t cbk )
```

This function initializes the PD phy registers.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>cbk</i>	The phy event handler callback.

Returns

ccg_status_t.

6.26.5.42 pd_phy_is_busy()

```
bool pd_phy_is_busy (
    uint8_t port )
```

This function checks if the PD phy is busy for the specified port.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

Returns true if the phy is busy, false otherwise.

6.26.5.43 pd_phy_load_msg()

```
bool pd_phy_load_msg (
    uint8_t port,
    sop_t sop,
    uint8_t retries,
    uint8_t dobj_count,
    uint32_t header,
    bool unchunked,
    uint32_t * buf )
```

This function loads the PD message in FIFO and configures the necessary registers.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>sop</i>	Sop type.
<i>retries</i>	Number of retries.
<i>dobj_count</i>	No of data objects(each 32 bit) in data
<i>header</i>	PD Header in lower 16 bits and optional unchunked extended header in upper 16 bits.
<i>unchunked</i>	Unchunked message if true.
<i>buf</i>	Pointer to message. Message buffer is an array of uint32_t. • buf[0]..buf[n] = Data, if data/extended message.

Returns

Returns true if successful, false otherwise.

6.26.5.44 pd_phy_refresh_roles()

```
void pd_phy_refresh_roles (
    uint8_t port )
```

This function configures the PD phy as per current port role / data role / contract status of the specified port. This API does not enable the receiver.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.45 pd_phy_reset_rx_tx_sm()

```
void pd_phy_reset_rx_tx_sm (
    uint8_t port )
```

Reset the PD receive and transmit state machines.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None

6.26.5.46 pd_phy_send_bist_cm2()

```
ccg_status_t pd_phy_send_bist_cm2 (
    uint8_t port )
```

This function starts transmission of BIST Carrier Mode 2.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

ccg_status_t

6.26.5.47 pd_phy_send_msg()

```
bool pd_phy_send_msg (
    uint8_t port )
```

This function starts the transmission of a message already loaded in FIFO.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

Returns true if successful, false otherwise.

6.26.5.48 pd_phy_send_reset()

```
ccg_status_t pd_phy_send_reset (
    uint8_t port,
    sop_t sop )
```

This function starts transmission of a cable reset or a hard reset as requested.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>sop</i>	Sop type = Cable Reset or Hard Reset.

Returns

ccg_status_t

6.26.5.49 pd_phy_wakeup()

```
bool pd_phy_wakeup (
    void )
```

This function configures the PD block on deepsleep exit.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

Returns true if successful, false otherwise.

6.26.5.50 pd_set_pfc_comp()

```
bool pd_set_pfc_comp (
    uint8_t port,
    uint32_t cur,
    filter_edge_detect_cfg_t edge,
    pd_cmp_cbk_t cbk )
```

Enables the PFC comparator.

Parameters

<i>port</i>	USB-PD port on which to enable comparator.
<i>cur</i>	Operating current in 10mA units.
<i>edge</i>	Which edge interrupt configure for.
<i>cbk</i>	Function to be called when interrupt happens.

Returns

State of the comparator after configuration.

6.26.5.51 pd_set_sr_comp()

```
bool pd_set_sr_comp (
    uint8_t port,
    uint32_t cur,
    filter_edge_detect_cfg_t edge,
    pd_cmp_cbk_t cbk )
```

Enables the SR comparator.

Parameters

<i>port</i>	USB-PD port on which to enable comparator.
<i>cur</i>	Operating current in 10mA units.
<i>edge</i>	Which edge interrupt configure for.
<i>cbk</i>	Function to be called when interrupt happens.

Returns

State of the comparator after configuration.

6.26.5.52 pd_stop_pfc_comp()

```
void pd_stop_pfc_comp (
    uint8_t port )
```

Disables PFC comparator on the specified port.

Parameters

<i>port</i>	USB-PD port.
-------------	--------------

Returns

None

6.26.5.53 pd_stop_sr_comp()

```
void pd_stop_sr_comp (
    uint8_t port )
```

Disables SR comparator on the specified port.

Parameters

<i>port</i>	USB-PD port.
-------------	--------------

Returns

None

6.26.5.54 pd_typec_dis_dpslp_rp()

```
void pd_typec_dis_dpslp_rp (
    uint8_t port )
```

This function disables deepsleep Rp.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.55 pd_typec_dis_rd()

```
void pd_typec_dis_rd (
    uint8_t port,
    uint8_t channel )
```

This function disables Rd.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

None.

6.26.5.56 pd_typec_dis_rp()

```
void pd_typec_dis_rp (
    uint8_t port,
    uint8_t channel )
```

This function disables Rp.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

None.

6.26.5.57 pd_typec_en_dpslp_rp()

```
void pd_typec_en_dpslp_rp (
    uint8_t port )
```

This function enables deepsleep Rp.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.58 pd_typec_en_rd()

```
void pd_typec_en_rd (
    uint8_t port,
    uint8_t channel )
```

This function enables Rd.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

None.

6.26.5.59 pd_typec_en_rp()

```
void pd_typec_en_rp (
    uint8_t port,
    uint8_t channel,
    rp_term_t rp_val )
```

This function configures and enables Rp.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.
<i>rp_val</i>	Rp value.

Returns

None.

6.26.5.60 pd_typec_get_cc_status()

```
cc_state_t pd_typec_get_cc_status (
    uint8_t port )
```

This function returns current CC status.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

cc_state_t

6.26.5.61 pd_typec_init()

```
ccg_status_t pd_typec_init (
    uint8_t port )
```

This function initializes the Type C registers in the PD block.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

ccg_status_t

6.26.5.62 pd_typec_rd_enable()

```
void pd_typec_rd_enable (
    uint8_t port )
```

This function enables the Rd termination without initializing the block completely.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

None.

6.26.5.63 pd_typec_set_polarity()

```
void pd_typec_set_polarity (
    uint8_t port,
    bool polarity )
```

This function sets the CC polarity for the receiver circuit.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

None.

6.26.5.64 pd_typec_snk_update_trim()

```
void pd_typec_snk_update_trim (
    uint8_t port )
```

This function updates the tx trim settings when in the sink role. It must be called when Rp is changed by the peer.

Parameters

<i>port</i>	Port index.
-------------	-------------

Returns

None.

6.26.5.65 pd_typec_start()

```
ccg_status_t pd_typec_start (
    uint8_t port )
```

This function starts the Type C line comparators. pdss_typec_init() should have been called before calls to this function.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

`ccg_status_t`.

6.26.5.66 pd_typec_stop()

```
ccg_status_t pd_typec_stop (
    uint8_t port )
```

: This function stops the Type-C line comparators.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
-------------	--

Returns

: ccg_status_t

6.26.5.67 pd_vconn_disable()

```
ccg_status_t pd_vconn_disable (
    uint8_t port,
    uint8_t channel )
```

This function turns off Vconn for the specified channel.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

ccg_status_t

6.26.5.68 pd_vconn_enable()

```
ccg_status_t pd_vconn_enable (
    uint8_t port,
    uint8_t channel )
```

This function turns on Vconn for the specified channel.

Parameters

<i>port</i>	Port index. Caller should ensure to provide only valid values.
<i>channel</i>	Channel index, where CC1 = 0, CC2 = 1.

Returns

ccg_status_t

6.26.5.69 soln_no_vsys_handler()

```
void soln_no_vsys_handler (
    void )
```

Solution level handler for cases where VSYS is not present. This is a user-defined handler function that needs to be defined in cases where the user wants to block device operation in bus powered mode. This function can be used to do any application level de-initialization to revert IOs to default state. Please note that PD disable should not be done here as removing Rd will cause a device reset.

Returns

None

6.26.5.70 soln_vsys_removal_handler()

```
void soln_vsys_removal_handler (
    void )
```

Solution level handler for VSYS removal event. This is a user-defined handler function that needs to be defined in cases where the user wants to block the device operation in bus powered mode. This function will be called in ISR context and hence should not block operation for a long time.

Returns

None

6.27 scb/i2c.h File Reference

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

Data Structures

- struct [i2c_scb_config_t](#)

Macros

- #define [I2C_BLOCK_COUNT](#) (4)
- #define [I2C_CLEAR_INTR_MASK](#) (0x00000000)
- #define [I2C_CLEAR_INTR_REQUEST_REG](#) (0xFFFFFFFFFu)
- #define [I2C_SCB_FIFO_SIZE](#) (8u)
- #define [I2C_SCB_RX_FIFO_SIZE](#) ([I2C_SCB_FIFO_SIZE](#))
- #define [I2C_SCB_TX_FIFO_SIZE](#) ([I2C_SCB_FIFO_SIZE](#))
- #define [I2C_SLAVE_ADDR_MASK_DEFAULT](#) (0xFE)
- #define [I2C_SLAVE_TIMER_BASE](#) (136u)
- #define [I2C_SLAVE_TIMER_PERIOD](#) (500)

Typedefs

- typedef bool(* [i2c_cb_fun_t](#)) ([i2c_cb_cmd_t](#) cmd, [i2c_scb_state_t](#) i2c_state, uint16_t count)

Enumerations

- enum `i2c_scb_state_t` {
 `I2C_SCB_STATE_DISABLED` = 0, `I2C_SCB_STATE_INIT`, `I2C_SCB_STATE_IDLE`, `I2C_SCB_STATE_PREAMBLE`, `I2C_SCB_STATE_READ`, `I2C_SCB_STATE_WRITE`, `I2C_SCB_STATE_CLK_STRETCH`, `I2C_SCB_STATE_ERROR`, `I2C_SCB_NUM_STATES` }
- enum `i2c_scb_mode_t` { `I2C_SCB_MODE_MASTER` = 0, `I2C_SCB_MODE_HPI`, `I2C_SCB_MODE_ALP_RIDGE` }
- enum `i2c_scb_clock_freq_t` { `I2C_SCB_CLOCK_FREQ_100_KHZ` = 0, `I2C_SCB_CLOCK_FREQ_400_KHZ`, `I2C_SCB_CLOCK_FREQ_1_MHZ` }
- enum `i2c_cb_cmd_t` {
 `I2C_CB_CMD_READ` = 0, `I2C_CB_CMD_WRITE`, `I2C_CB_CMD_XFER_END`, `I2C_CB_CMD_TIMEOUT`, `I2C_CB_SLAVE_ADDR_MATCH` }

Functions

- void `i2c_scb_init` (uint8_t `scb_index`, `i2c_scb_mode_t` `mode`, `i2c_scb_clock_freq_t` `clock_freq`, uint8_t `slave_addr`, uint8_t `slave_mask`, `i2c_cb_fun_t` `cb_fun_ptr`, uint8_t *`scratch_buffer`, uint16_t `scratch_buffer_size`)
- void `i2c_scb_deinit` (uint8_t `scb_index`)
- void `i2c_scb_write` (uint8_t `scb_index`, uint8_t *`source_ptr`, uint8_t `size`, uint8_t *`count`)
- void `i2c_reset` (uint8_t `scb_index`)
- void `i2c_slave_ack_ctrl` (uint8_t `scb_index`, bool `enable`)
- bool `i2c_scb_is_idle` (uint8_t `scb_index`)
- void `i2c_scb_enable_wakeup` (uint8_t `scb_index`)

6.27.1 Detailed Description

I2C slave driver header file.

6.27.2 Macro Definition Documentation

6.27.2.1 I2C_BLOCK_COUNT

```
#define I2C_BLOCK_COUNT (4)
```

Number of I2C blocks supported by the device.

The CCG3 and CCG4 device families support 4 SCB blocks which can be used for I2C functionality. CCG3PA, CCG3PA2 supports two SCB blocks. If the target device using this stack supports a different number of SCBs, this macro value will need to be updated.

6.27.3 Enumeration Type Documentation

6.27.3.1 i2c_cb_cmd_t

```
enum i2c_cb_cmd_t
```

Type of I2C operation being notified through a callback function.

Enumerator

I2C_CB_CMD_READ	Read command from master.
I2C_CB_CMD_WRITE	Write command from master.
I2C_CB_CMD_XFER_END	End of read transfer: STOP condition signalled by master.
I2C_CB_CMD_TIMEOUT	Timeout on I2C operation.
I2C_CB_SLAVE_ADDR_MATCH	I2C slave address match detected.

6.27.3.2 i2c_scb_clock_freq_t

```
enum i2c_scb_clock_freq_t
```

List of possible I2C bus bit rates.

Enumerator

I2C_SCB_CLOCK_FREQ_100_KHZ	100 KHz operation.
I2C_SCB_CLOCK_FREQ_400_KHZ	400 KHz operation.
I2C_SCB_CLOCK_FREQ_1_MHZ	1 MHz operation.

6.27.3.3 i2c_scb_mode_t

```
enum i2c_scb_mode_t
```

List of possible I2C block operating modes.

This type lists the possible I2C block operating modes. Multiple slave modes are defined in case there is any mode specific handling required for any of them.

Note: Master mode is currently not supported by the driver. Note: The HPI and ALP_RIDGE mode handling is equivalent.

Enumerator

I2C_SCB_MODE_MASTER	Master mode: Can be used for mux control. Not supported as of now.
I2C_SCB_MODE_HPI	Slave mode: Used for HPI interface.
I2C_SCB_MODE_ALP_RIDGE	Slave mode: Used for Alpine Ridge interface.

6.27.3.4 i2c_scb_state_t

```
enum i2c_scb_state_t
```

List of possible I2C block states.

The I2C driver implements a state machine which ensures that read/write transfers are responded to correctly without any corruption. This data type lists the possible states in this slave mode state machine.

Enumerator

I2C_SCB_STATE_DISABLED	I2C interface is disabled.
------------------------	----------------------------

Enumerator

I2C_SCB_STATE_INIT	Interface initialized and waiting to be enabled.
I2C_SCB_STATE_IDLE	Interface ready and waiting for preamble from the master.
I2C_SCB_STATE_PREAMBLE	Preamble phase is in progress.
I2C_SCB_STATE_READ	I2C read operation is in progress.
I2C_SCB_STATE_WRITE	I2C write operation is in progress.
I2C_SCB_STATE_CLK_STRETCH	Drive is stretching I2C clock to delay operation.
I2C_SCB_STATE_ERROR	Error state: transaction error detected.
I2C_SCB_NUM_STATES	Last state ID: not used.

6.27.4 Function Documentation

6.27.4.1 i2c_reset()

```
void i2c_reset (
    uint8_t scb_index )
```

Reset the I2C block specified.

This function resets the I2C block in response to an error or explicit request from protocol layer.

Parameters

<i>scb_index</i>	SCB ID to be reset.
------------------	---------------------

Returns

None

6.27.4.2 i2c_scb_deinit()

```
void i2c_scb_deinit (
    uint8_t scb_index )
```

De-initialize a previously initialized SCB block.

Parameters

<i>scb_index</i>	SCB index to be disabled.
------------------	---------------------------

Returns

None

6.27.4.3 i2c_scb_enable_wakeup()

```
void i2c_scb_enable_wakeup (
    uint8_t scb_index )
```

Enable deep-sleep wakeup due to address match on the specified SCB block.

Parameters

<i>scb_index</i>	SCB ID to be configured as deep-sleep wakeup trigger.
------------------	---

Returns

None

6.27.4.4 i2c_scb_init()

```
void i2c_scb_init (
    uint8_t scb_index,
    i2c_scb_mode_t mode,
    i2c_scb_clock_freq_t clock_freq,
    uint8_t slave_addr,
    uint8_t slave_mask,
    i2c_cb_fun_t cb_fun_ptr,
    uint8_t * scratch_buffer,
    uint16_t scratch_buffer_size )
```

Configure one of the I2C blocks as required.

This API is used to enable and configure one of the I2C blocks for driver operation. Only I2C slave operation is currently supported by the driver as of now.

The I2C driver is agnostic of the actual data transfer protocol. It reads all data written by the master into a receive buffer provided by the protocol layer. A callback function is used to notify the protocol layer when the write is complete. The receive buffer provided should be big enough to hold the maximum amount of data that the master may provide in a write operation. If the write contains more data than the buffer can hold, the I2C driver will NAK the transaction.

Read requests from the I2C master are automatically delayed by clock stretching. A callback function is used to notify the protocol layer that the master is waiting for data. The i2c_scb_write function can be used by the protocol layer to write data into the transmit FIFO in response to the read request.

All I2C driver events are generated from interrupt context, and are expected to be handled with care. The protocol layer should defer any long operations to a non-interrupt context.

Parameters

<i>scb_index</i>	SCB index being configured for I2C operation.
<i>mode</i>	Desired mode of operation.
<i>clock_freq</i>	Desired I2C clock frequency.
<i>slave_addr</i>	Device address to be used in case of slave operation.
<i>slave_mask</i>	Mask to be applied on for slave address matching.
<i>cb_fun_ptr</i>	Callback function to be called for event notification.
<i>scratch_buffer</i>	Receive buffer used to hold written by master.
<i>scratch_buffer_size</i>	Size of the receive buffer in bytes.

Returns

None

6.27.4.5 i2c_scb_is_idle()

```
bool i2c_scb_is_idle (
    uint8_t scb_index )
```

Check whether the I2C block is idle.

This function checks whether the specified I2C block is idle. This check should be performed before the device enters deep sleep. Deep sleep entry should be avoided if this function returns false.

Parameters

<i>scb_index</i>	SCB ID to be checked.
------------------	-----------------------

Returns

true if the I2C block is idle, false otherwise.

6.27.4.6 i2c_scb_write()

```
void i2c_scb_write (
    uint8_t scb_index,
    uint8_t * source_ptr,
    uint8_t size,
    uint8_t * count )
```

Write data into the transmit FIFO associated with the I2C block.

This function is used by the protocol layer to write data into the I2C transmit FIFO in response to a master read operation.

Parameters

<i>scb_index</i>	SCB ID to do the I2C transfer through.
<i>source_ptr</i>	Pointer to buffer containing the data to be written.
<i>size</i>	Size of the data buffer. Maximum amount of data that may be written.
<i>count</i>	Return parameter through which the actual write size is returned.

Returns

None

6.27.4.7 i2c_slave_ack_ctrl()

```
void i2c_slave_ack_ctrl (
    uint8_t scb_index,
    bool enable )
```

Enable/Disable the I2C slave acknowledgement.

This function enables/disables the slave address ACK from the I2C block. The protocol layer can disable the address ACK to hold off data transfers when it is not ready to respond to the master.

Parameters

<code>scb_index</code>	SCB ID to configure.
<code>enable</code>	Whether to enable or disable the auto slave address acknowledgement.

Returns

None

6.28 system/boot.h File Reference

```
#include "config.h"
#include "status.h"
#include "system.h"
```

Data Structures

- union `fw_img_status_t`
- struct `fw_img_status_t::fw_mode_reason_t`
- struct `sys_fw_metadata_t`

Macros

- #define `CY_PD_IMG1_FW_STATUS_BIT_MASK` (0x08)
- #define `CCG_BOOT_MODE_RQT_SIG` (0x424C)
- #define `CCG_FW1_BOOT_RQT_SIG` (0x4231)
- #define `CCG_FW2_BOOT_RQT_SIG` (0x4232)
- #define `CCG_FW_METADATA_BOOTSEQ_OFFSET` (0x1C)
- #define `CCG_BL_WAIT_DEFAULT` (50)
- #define `CCG_BL_WAIT_MINIMUM` (20)
- #define `CCG_BL_WAIT_MAXIMUM` (1000)
- #define `CCG_FWMETA_APPID_WAIT_DEF` (0xFFFFu)
- #define `CCG_FWMETA_APPID_WAIT_0` (0x4359)
- #define `CONFIGTABLE_SIGNATURE` (0x4359)
- #define `CONFIGTABLE_SIZE_OFFSET` (6)
- #define `CONFIGTABLE_CHECKSUM_OFFSET` (8)
- #define `CONFIGTABLE_CHECKSUM_START` (10)

Functions

- `ccg_status_t boot_validate_configtable (uint8_t *table_p)`
- `ccg_status_t boot_validate_firmware (sys_fw_metadata_t *fw_metadata)`
- `ccg_status_t boot_handle_validate_firmware_cmd (sys_fw_mode_t fw_mode)`
- `uint16_t boot_get_wait_time (void)`
- `bool boot_start (void)`
- `void boot_check_for_valid_firmware (void)`

- `fw_img_status_t get_boot_mode_reason (void)`
- `void boot_jump_to_fw (void)`
- `uint32_t boot_get_boot_seq (uint8_t fwid)`
- `void boot_update_fw_status (void)`

Variables

- `sys_fw_metadata_t * gl_img1_fw_metadata`
- `sys_fw_metadata_t * gl_img2_fw_metadata`
- `sys_fw_metadata_t * gl_img1_fw_pseudo_metadata`
- `sys_fw_metadata_t * gl_img2_fw_pseudo_metadata`
- `fw_img_status_t gl_img_status`

6.28.1 Detailed Description

Bootloader support header file.

6.28.2 Function Documentation

6.28.2.1 boot_check_for_valid_fw()

```
void boot_check_for_valid_fw (
    void )
```

Check for the presence of alternate firmware waiting to be validated.

This function checks whether the CCGx device flash contains an alternate firmware binary which is waiting to be validated. This can happen if a firmware update happened during the last power up of the device, and the binary is yet to be validated and made active. The active firmware will make use of the pseudo metadata in flash to identify the alternate firmware, validate it and activate it by updating the actual firmware metadata.

Please refer to the CCGx boot sequence description in the firmware guide for a more detailed description of the boot procedure.

Returns

NONE

6.28.2.2 boot_get_boot_seq()

```
uint32_t boot_get_boot_seq (
    uint8_t fwid )
```

Get the boot sequence number value for the specified firmware image.

A boot sequence number field stored in the firmware metadata is used by the CCGx boot-loader to identify the firmware binary to be loaded. This function retrieves the sequence number associated with the specified firmware binary.

Parameters

<code>fwid</code>	Firmware id whose sequence number is to be retrieved. 1 for FW1 and 2 for FW2.
-------------------	--

Returns

Boot sequence number value if the firmware is valid, 0 otherwise.

6.28.2.3 boot_get_wait_time()

```
uint16_t boot_get_wait_time (
    void )
```

Returns the boot-wait delay configured for the application.

This function identifies the boot-wait delay required by checking the firmware metadata.

Returns

Boot-wait delay in milliseconds.

6.28.2.4 boot_handle_validate_fw_cmd()

```
ccg_status_t boot_handle_validate_fw_cmd (
    sys_fw_mode_t fw_mode )
```

Handles the VALIDATE_FW command from HPI or UVDM.

This API handles the VALIDATE_FW command received through the HPI or UVDM interfaces.

Parameters

<i>fw_mode</i>	Firmware binary id: 1 for FW1 and 2 for FW2.
----------------	--

Returns

Status code indicating the validity of the firmware.

6.28.2.5 boot_jump_to_fw()

```
void boot_jump_to_fw (
    void )
```

Transfer control to the firmware binary identified by boot_start.

This function is only used by the CCGx boot-loader. This transfers control to the firmware binary selected as the boot target by the boot_start function. This is expected to be called after the boot-wait window has elapsed.

Returns

None

6.28.2.6 boot_start()

```
bool boot_start (
    void )
```

Identify the firmware binary to be loaded.

This function is only used in the CCGx boot-loader, and implements the main start-up logic of the boot-loader. The function validates the two firmware binaries in device flash, and identifies the binary to be loaded. If neither binary is valid, the function returns false notifying the caller to continue in boot-loader mode.

Returns

true if firmware load is allowed, false otherwise.

6.28.2.7 boot_update_fw_status()

```
void boot_update_fw_status (
    void )
```

Function to validate firmware images and update image status.

This function is used to validate the firmware images in the CCG device flash and update their status in the image status / boot-mode reason field.

Returns

None

6.28.2.8 boot_validate_configtable()

```
ccg_status_t boot_validate_configtable (
    uint8_t * table_p )
```

Validate the configuration table specified.

Each copy of CCGx firmware on the device flash contains an embedded configuration table that defines the runtime behaviour of the CCGx device. This function checks whether the configuration table located at the specified location is valid (has valid signature and checksum).

Parameters

<i>table_p</i>	Pointer to the configuration table to be validated.
----------------	---

Returns

CCG_STAT_SUCCESS if the table is valid, CCG_STAT_FAILURE otherwise.

6.28.2.9 boot_validate_fw()

```
ccg_status_t boot_validate_fw (
    sys_fw_metadata_t * fw_metadata )
```

Validate the firmware image associated with the given metadata.

This function validates the firmware binary associated with the metadata specified in the fw_metadata parameter. The validity check includes checks for signature, location, size and checksum. This function internally performs validation of the embedded configuration table using the boot_validate_configtable function.

Parameters

<code>fw_metadata</code>	Pointer to metadata table of the FW which has to be validated.
--------------------------	--

Returns

`CCG_STAT_SUCCESS` if the firmware is valid, `CCG_STAT_FAILURE` otherwise.

6.28.2.10 `get_boot_mode_reason()`

```
fw_img_status_t get_boot_mode_reason (
    void )
```

Returns a bitmap containing the reason for boot mode.

This function returns the bitmap value that is to be stored in the `BOOT_MODE_REASON` HPI register, which identifies the validity of the two firmware binaries. The validation of the firmware is expected to have been completed earlier through the `boot_start` function. This function only retrieves the status stored during the validation procedure.

See also

[fw_img_status_t](#)

Returns

Boot mode reason bitmap.

6.29 system/ccgx_api_desc.h File Reference

6.29.1 Detailed Description

CCGx Firmware API Reference Guide Introduction.

6.30 system/ccgx_version.h File Reference

Macros

- #define `FW_MAJOR_VERSION` (3)
- #define `FW_MINOR_VERSION` (2)
- #define `FW_PATCH_VERSION` (1)
- #define `FW_BUILD_NUMBER` (1658)
- #define `FW_BASE_VERSION`

6.30.1 Detailed Description

This file defines the base firmware stack version for CCGx.

6.30.2 Macro Definition Documentation

6.30.2.1 FW_BASE_VERSION

```
#define FW_BASE_VERSION
Value:
((FW_MAJOR_VERSION << 28) | (FW_MINOR_VERSION << 24) |      \
(FW_PATCH_VERSION << 16) | (FW_BUILD_NUMBER))
```

Composite firmware stack version value.

Composite firmware stack version value. This is a 4 byte value with the following format: Bytes 1-0: Build number
Byte 2: Patch version Byte 3 (Bits 0:3): Minor Version Byte 3 (Bits 4:7): Major Version

6.31 system/flash.h File Reference

```
#include "stdint.h"
#include "stdbool.h"
#include "config.h"
#include "status.h"
```

Macros

- `#define SROM_API_PARAM_SIZE (8u)`

Typedefs

- `typedef void(* flash_cbk_t) (flash_write_status_t status)`

Enumerations

- `enum flash_app_priority_t { FLASH_APP_PRIORITY_DEFAULT = 0, FLASH_APP_PRIORITY_IMAGE_1, FLASH_APP_PRIORITY_IMAGE_2 }`
- `enum flash_interface_t { FLASH_IF_HPI = 0, FLASH_IF_UVDM, FLASH_IF_USB_HID, FLASH_IF_IECS_UART }`
- `enum flash_write_status_t { FLASH_WRITE_COMPLETE, FLASH_WRITE_ABORTED, FLASH_WRITE_COMPLETE_AND_A FLASH_WRITE_IN_PROGRESS }`

Functions

- `void flash_enter_mode (bool is_enable, flash_interface_t mode, bool data_in_place)`
- `bool flash_access_get_status (uint8_t modes)`
- `void flash_set_access_limits (uint16_t start_row, uint16_t last_row, uint16_t md_row, uint16_t bl_last_row)`
- `ccg_status_t flash_row_clear (uint16_t row_num)`
- `ccg_status_t flash_row_write (uint16_t row_num, uint8_t *data, flash_cbk_t cbk)`
- `ccg_status_t flash_row_read (uint16_t row_num, uint8_t *data)`
- `ccg_status_t flash_set_app_priority (flash_app_priority_t app_priority)`

6.31.1 Detailed Description

Flash command handler header file.

6.31.2 Typedef Documentation

6.31.2.1 flash_cbk_t

```
flash_cbk_t
```

Non-blocking flash write row callback function type.

The CCG3 device supports non-blocking flash update operations, so that the firmware can perform other operations while a flash row write is in progress. The completion of the flash row write is notified through a callback function. This type represents the function type which can be registered as a callback for notification of non-blocking flash operations.

6.31.3 Enumeration Type Documentation

6.31.3.1 flash_app_priority_t

```
enum flash_app_priority_t
```

Enumeration of app priority values.

App priority is a debug feature that allows the CCG bootloader to prioritize one copy of the firmware over the other. The default behaviour is for both firmware binaries to have the same priority so that the more recently updated firmware binary gets loaded. The priority scheme can be updated to allow FW1 or FW2 to always be loaded for debugging purposes.

Enumerator

<code>FLASH_APP_PRIORITY_DEFAULT</code>	Default. Latest image gets priority.
<code>FLASH_APP_PRIORITY_IMAGE_1</code>	Image-1 gets priority over image-2.
<code>FLASH_APP_PRIORITY_IMAGE_2</code>	Image-2 gets priority over image-1.

6.31.3.2 flash_interface_t

```
enum flash_interface_t
```

List of supported flash update interfaces.

CCG supports multiple flash update interfaces such as HPI, UVDM, USB etc. depending on the application. This enumerated type lists the supported flash update interfaces.

Enumerator

<code>FLASH_IF_HPI</code>	HPI based flash update interface.
<code>FLASH_IF_UVDM</code>	UVDM based flash update interface.
<code>FLASH_IF_USB_HID</code>	USB flash update interface.
<code>FLASH_IF_IECS_UART</code>	IECS (UART) flash update interface.

6.31.3.3 flash_write_status_t

```
enum flash_write_status_t
```

List of possible status codes for non blocking flash write operation.

Enumerator

FLASH_WRITE_COMPLETE	Flash Write successfully completed.
FLASH_WRITE_ABORTED	Flash Write aborted.
FLASH_WRITE_COMPLETE_AND_ABORTED	Flash Write completed with an abort request.
FLASH_WRITE_IN_PROGRESS	Flash Write is active.

6.31.4 Function Documentation

6.31.4.1 flash_access_get_status()

```
bool flash_access_get_status (
    uint8_t modes )
```

Check whether flashing mode has been entered.

This function checks whether flashing mode has currently been entered by a different interface.

Parameters

<i>modes</i>	Bitmap containing flashing interfaces to be checked.
--------------	--

Returns

Returns true if flashing mode has been entered using any of the interfaces listed in modes, false otherwise.

6.31.4.2 flash_enter_mode()

```
void flash_enter_mode (
    bool is_enable,
    flash_interface_t mode,
    bool data_in_place )
```

Handle ENTER_FLASHING_MODE Command.

This function notifies the CCG stack that flash read/write is being enabled by the application. By default, C \leftarrow CG firmware disallows all flash read/write operations. Flash access is only allowed after flashing mode has been explicitly enabled through user command.

Note: FW update interface is allowed to Enable Flashing mode only once in one session. This is to ensure that multiple flashing interfaces are not active simultaneously. Each FW update interface is expected to take care of this. Once Flash access is complete, this API should be used to exit FW Update interface.

Parameters

<i>is_enable</i>	Enable/Disable Flashing Mode
<i>mode</i>	Flash update interface to be used.
<i>data_in_place</i>	Specifies whether the flash write data buffer can be used in place as SROM API parameter buffer.

Returns

None

6.31.4.3 flash_row_clear()

```
ccg_status_t flash_row_clear (
    uint16_t row_num )
```

Erase the contents of the specified flash row.

This API erases the contents of the specified flash row by filling it with zeroes. Please note that this API will only work if flashing mode is enabled and the selected row is within the range of write enabled rows.

Parameters

<i>row_num</i>	Row number to be erased.
----------------	--------------------------

Returns

Status of row erase operation.

6.31.4.4 flash_row_read()

```
ccg_status_t flash_row_read (
    uint16_t row_num,
    uint8_t * data )
```

Read the contents of the specified flash row.

This API handles the flash read row operation. The contents of the flash row are copied into the specified data buffer, if flashing mode has been entered and the row_num is part of the readable range of memory.

Parameters

<i>row_num</i>	Flash row to be read.
<i>data</i>	Buffer to read the flash row content into.

Returns

Status of the flash read. CCG_STAT_SUCCESS or appropriate error code.

6.31.4.5 flash_row_write()

```
ccg_status_t flash_row_write (
    uint16_t row_num,
    uint8_t * data,
    flash_cbk_t cbk )
```

Write the given data to the specified flash row.

This API handles the flash write row operation. The contents from the data buffer is written to the row_num flash row. The access rules for the flash row as the same as for the flash_row_clear API.

For non-blocking write row operation, the API returns as soon as the row update is started. The stack takes care of executing all of the steps across multiple resume interrupts; and the callback is called at the end of the process.

Parameters

<i>row_num</i>	Flash row to be updated.
<i>data</i>	Buffer containing data to be written to the flash row.
<i>cbk</i>	Callback function to be called at the end of non-blocking flash write.

Returns

Status of the flash write. CCG_STAT_SUCCESS or appropriate error code.

6.31.4.6 flash_set_access_limits()

```
void flash_set_access_limits (
    uint16_t start_row,
    uint16_t last_row,
    uint16_t md_row,
    uint16_t bl_last_row )
```

Set limits to the flash rows that can be accessed.

The CCG stack has been designed to support fail-safe firmware upgrades using a pair of firmware binaries that can mutually update each other. This scheme can only be robust if the currently active firmware binary can effectively protect itself by not allowing access to any of its own flash rows.

This function is used to specify the list of flash rows that can safely be accessed by the currently active firmware binary. This function should only be used with parameters derived based on the binary locations identified from firmware metadata. Incorrect usage of this API can cause the device to stop responding during a flash update operation.

This API must be invoked as part of initialization before the [flash_row_write\(\)](#) and [flash_row_read\(\)](#) functions can be called. By default, no flash row can be read or written to.

Parameters

<i>start_row</i>	The lowest row number that can be written.
<i>last_row</i>	The highest row number that can be written.
<i>md_row</i>	Row containing metadata for the alternate firmware.
<i>bl_last_row</i>	Last bootloader row. Rows above this can be read.

Returns

None

6.31.4.7 flash_set_app_priority()

```
ccg_status_t flash_set_app_priority (
    flash_app_priority_t app_priority )
```

Updates the app boot priority flag.

This function is used to set the app priority field to override the default FW selection algorithm.

Parameters

<i>app_priority</i>	Desired boot priority setting.
---------------------	--------------------------------

Returns

Status code of the priority update.

6.32 system/gpio.h File Reference

```
#include "config.h"
#include "stdint.h"
#include "stdbool.h"
```

Macros

- #define **GPIO_DM_FIELD_SIZE** (3u)
- #define **GPIO_DM_FIELD_MASK** (7u)
- #define **GPIO_INT_FIELD_MASK** (3u)
- #define **HSIOM_FIELD_SHIFT** (2u)
- #define **GPIO_PORT0_INTR_NO** (0u)
- #define **GPIO_PORT1_INTR_NO** (1u)
- #define **GPIO_PORT2_INTR_NO** (2u)
- #define **GPIO_PORT3_INTR_NO** (3u)
- #define **GPIO_PORT4_INTR_NO** (4u)
- #define **GPIO_PORT5_INTR_NO** (5u)

Enumerations

- enum **gpio_port_pin_t** {
 GPIO_PORT_0_PIN_0 = 0x00, **GPIO_PORT_0_PIN_1**, **GPIO_PORT_0_PIN_2**, **GPIO_PORT_0_PIN_3**,
GPIO_PORT_0_PIN_4, **GPIO_PORT_0_PIN_5**, **GPIO_PORT_0_PIN_6**, **GPIO_PORT_0_PIN_7**,
GPIO_PORT_1_PIN_0 = 0x10, **GPIO_PORT_1_PIN_1**, **GPIO_PORT_1_PIN_2**, **GPIO_PORT_1_PIN_3**,
GPIO_PORT_1_PIN_4, **GPIO_PORT_1_PIN_5**, **GPIO_PORT_1_PIN_6**, **GPIO_PORT_1_PIN_7**,
GPIO_PORT_2_PIN_0 = 0x20, **GPIO_PORT_2_PIN_1**, **GPIO_PORT_2_PIN_2**, **GPIO_PORT_2_PIN_3**,
GPIO_PORT_2_PIN_4, **GPIO_PORT_2_PIN_5**, **GPIO_PORT_2_PIN_6**, **GPIO_PORT_2_PIN_7**,
GPIO_PORT_3_PIN_0 = 0x30, **GPIO_PORT_3_PIN_1**, **GPIO_PORT_3_PIN_2**, **GPIO_PORT_3_PIN_3**,
GPIO_PORT_3_PIN_4, **GPIO_PORT_3_PIN_5**, **GPIO_PORT_3_PIN_6**, **GPIO_PORT_3_PIN_7**,
 }

- ```

GPIO_PORT_4_PIN_0 = 0x40, GPIO_PORT_4_PIN_1, GPIO_PORT_4_PIN_2, GPIO_PORT_4_PIN_3,
GPIO_PORT_4_PIN_4, GPIO_PORT_4_PIN_5, GPIO_PORT_4_PIN_6, GPIO_PORT_4_PIN_7,
GPIO_PORT_5_PIN_0 = 0x50, GPIO_PORT_5_PIN_1, GPIO_PORT_5_PIN_2, GPIO_PORT_5_PIN_3,
GPIO_PORT_5_PIN_4, GPIO_PORT_5_PIN_5, GPIO_PORT_5_PIN_6, GPIO_PORT_5_PIN_7,
GPIO_PORT_6_PIN_0 = 0x60, GPIO_PORT_6_PIN_1 }

• enum gpio_dm_t {
 GPIO_DM_HIZ_ANALOG = 0, GPIO_DM_HIZ_DIGITAL, GPIO_DM_RES_UP, GPIO_DM_RES_DWN,
 GPIO_DM_OD_LOW, GPIO_DM_OD_HIGH, GPIO_DM_STRONG, GPIO_DM_RES_UPDOWN }

• enum gpio_intr_t { GPIO_INTR_DISABLE = 0, GPIO_INTR_RISING, GPIO_INTR_FALLING, GPIO_INTR_BOTH }

• enum hsiom_mode_t { HSIOM_MODE_GPIO = 0 }

```

## Functions

- void [gpio\\_set\\_value](#) ([gpio\\_port\\_pin\\_t](#) port\_pin, bool value)
- bool [gpio\\_read\\_value](#) ([gpio\\_port\\_pin\\_t](#) port\_pin)
- void [gpio\\_set\\_drv\\_mode](#) ([gpio\\_port\\_pin\\_t](#) port\_pin, [gpio\\_dm\\_t](#) drv\_mode)
- void [gpio\\_int\\_set\\_config](#) ([gpio\\_port\\_pin\\_t](#) port\_pin, uint8\_t int\_mode)
- void [hsiom\\_set\\_config](#) ([gpio\\_port\\_pin\\_t](#) port\_pin, [hsiom\\_mode\\_t](#) hsiom\_mode)
- void [gpio\\_hsiom\\_set\\_config](#) ([gpio\\_port\\_pin\\_t](#) port\_pin, [hsiom\\_mode\\_t](#) hsiom\_mode, [gpio\\_dm\\_t](#) drv\_mode, bool value)
- void [gpio\\_set\\_lvttl\\_mode](#) (uint8\_t port)
- bool [gpio\\_get\\_intr](#) ([gpio\\_port\\_pin\\_t](#) port\_pin)
- void [gpio\\_clear\\_intr](#) ([gpio\\_port\\_pin\\_t](#) port\_pin)

### 6.32.1 Detailed Description

GPIO and IO mapping control functions.

### 6.32.2 Enumeration Type Documentation

#### 6.32.2.1 [gpio\\_dm\\_t](#)

enum [gpio\\_dm\\_t](#)

Various GPIO drive modes supported by the CCGx devices.

This enumeration lists the various drive modes supported by CCGx IOs which are configured as GPIOs. Please refer to [hsiom\\_mode\\_t](#) for the IO mapping settings available for each CCGx IO.

See also

[hsiom\\_mode\\_t](#)

#### Enumerator

|                                     |                                           |
|-------------------------------------|-------------------------------------------|
| <a href="#">GPIO_DM_HIZ_ANALOG</a>  | Output buffer off (HiZ), input buffer off |
| <a href="#">GPIO_DM_HIZ_DIGITAL</a> | Output buffer off (HiZ), input buffer on  |
| <a href="#">GPIO_DM_RES_UP</a>      | Resistive pull-up                         |
| <a href="#">GPIO_DM_RES_DWN</a>     | Resistive pull-down                       |
| <a href="#">GPIO_DM_OD_LOW</a>      | Drive low and HZI for high                |
| <a href="#">GPIO_DM_OD_HIGH</a>     | Drive high and HZI for low                |

## Enumerator

|                    |                                 |
|--------------------|---------------------------------|
| GPIO_DM_STRONG     | Strong low and high drive       |
| GPIO_DM_RES_UPDOWN | Resistive pull-up and pull-down |

## 6.32.2.2 gpio\_intr\_t

```
enum gpio_intr_t
```

Various GPIO interrupt modes supported by the device.

## Enumerator

|                   |                                             |
|-------------------|---------------------------------------------|
| GPIO_INTR_DISABLE | GPIO interrupt disabled.                    |
| GPIO_INTR_RISING  | Interrupt on rising edge of input.          |
| GPIO_INTR_FALLING | Interrupt on falling edge of input.         |
| GPIO_INTR_BOTH    | Interrupt on both rising and falling edges. |

## 6.32.2.3 gpio\_port\_pin\_t

```
enum gpio_port_pin_t
```

List of pins supported on CCGx devices.

This enumeration lists the port and pin assignment for the pins available on the CCGx USB-PD controllers. This is a superset of all pins supported across all of the devices in the family, and includes pins such as CC1/CC2 which have fixed functionality.

Please refer to the respective device datasheets to identify the pins that can be used as GPIOs.

## Enumerator

|                   |                      |
|-------------------|----------------------|
| GPIO_PORT_0_PIN_0 | P0.0: Port 0, Pin 0. |
| GPIO_PORT_0_PIN_1 | P0.1: Port 0, Pin 1. |
| GPIO_PORT_0_PIN_2 | P0.2: Port 0, Pin 2. |
| GPIO_PORT_0_PIN_3 | P0.3: Port 0, Pin 3. |
| GPIO_PORT_0_PIN_4 | P0.4: Port 0, Pin 4. |
| GPIO_PORT_0_PIN_5 | P0.5: Port 0, Pin 5. |
| GPIO_PORT_0_PIN_6 | P0.6: Port 0, Pin 6. |
| GPIO_PORT_0_PIN_7 | P0.7: Port 0, Pin 7. |
| GPIO_PORT_1_PIN_0 | P1.0: Port 1, Pin 0. |
| GPIO_PORT_1_PIN_1 | P1.1: Port 1, Pin 1. |

## Enumerator

|                                |                      |
|--------------------------------|----------------------|
| <code>GPIO_PORT_1_PIN_2</code> | P1.2: Port 1, Pin 2. |
| <code>GPIO_PORT_1_PIN_3</code> | P1.3: Port 1, Pin 3. |
| <code>GPIO_PORT_1_PIN_4</code> | P1.4: Port 1, Pin 4. |
| <code>GPIO_PORT_1_PIN_5</code> | P1.5: Port 1, Pin 5. |
| <code>GPIO_PORT_1_PIN_6</code> | P1.6: Port 1, Pin 6. |
| <code>GPIO_PORT_1_PIN_7</code> | P1.7: Port 1, Pin 7. |
| <code>GPIO_PORT_2_PIN_0</code> | P2.0: Port 2, Pin 0. |
| <code>GPIO_PORT_2_PIN_1</code> | P2.1: Port 2, Pin 1. |
| <code>GPIO_PORT_2_PIN_2</code> | P2.2: Port 2, Pin 2. |
| <code>GPIO_PORT_2_PIN_3</code> | P2.3: Port 2, Pin 3. |
| <code>GPIO_PORT_2_PIN_4</code> | P2.4: Port 2, Pin 4. |
| <code>GPIO_PORT_2_PIN_5</code> | P2.5: Port 2, Pin 5. |
| <code>GPIO_PORT_2_PIN_6</code> | P2.6: Port 2, Pin 6. |
| <code>GPIO_PORT_2_PIN_7</code> | P2.7: Port 2, Pin 7. |
| <code>GPIO_PORT_3_PIN_0</code> | P3.0: Port 3, Pin 0. |
| <code>GPIO_PORT_3_PIN_1</code> | P3.1: Port 3, Pin 1. |
| <code>GPIO_PORT_3_PIN_2</code> | P3.2: Port 3, Pin 2. |
| <code>GPIO_PORT_3_PIN_3</code> | P3.3: Port 3, Pin 3. |
| <code>GPIO_PORT_3_PIN_4</code> | P3.4: Port 3, Pin 4. |
| <code>GPIO_PORT_3_PIN_5</code> | P3.5: Port 3, Pin 5. |
| <code>GPIO_PORT_3_PIN_6</code> | P3.6: Port 3, Pin 6. |
| <code>GPIO_PORT_3_PIN_7</code> | P3.7: Port 3, Pin 7. |
| <code>GPIO_PORT_4_PIN_0</code> | P4.0: Port 4, Pin 0. |
| <code>GPIO_PORT_4_PIN_1</code> | P4.1: Port 4, Pin 1. |
| <code>GPIO_PORT_4_PIN_2</code> | P4.2: Port 4, Pin 2. |
| <code>GPIO_PORT_4_PIN_3</code> | P4.3: Port 4, Pin 3. |

## Enumerator

|                   |                      |
|-------------------|----------------------|
| GPIO_PORT_4_PIN_4 | P4.4: Port 4, Pin 4. |
| GPIO_PORT_4_PIN_5 | P4.5: Port 4, Pin 5. |
| GPIO_PORT_4_PIN_6 | P4.6: Port 4, Pin 6. |
| GPIO_PORT_4_PIN_7 | P4.7: Port 4, Pin 7. |
| GPIO_PORT_5_PIN_0 | P5.0: Port 5, Pin 0. |
| GPIO_PORT_5_PIN_1 | P5.1: Port 5, Pin 1. |
| GPIO_PORT_5_PIN_2 | P5.2: Port 5, Pin 2. |
| GPIO_PORT_5_PIN_3 | P5.3: Port 5, Pin 3. |
| GPIO_PORT_5_PIN_4 | P5.4: Port 5, Pin 4. |
| GPIO_PORT_5_PIN_5 | P5.5: Port 5, Pin 5. |
| GPIO_PORT_5_PIN_6 | P5.6: Port 5, Pin 6. |
| GPIO_PORT_5_PIN_7 | P5.7: Port 5, Pin 7. |
| GPIO_PORT_6_PIN_0 | P6.0: Port 6, Pin 0. |
| GPIO_PORT_6_PIN_1 | P6.1: Port 6, Pin 1. |

## 6.32.2.4 hsiom\_mode\_t

```
enum hsiom_mode_t
```

Various IO matrix configuration modes.

Most of the IOs on CCGx devices & DMC (Dock Management Controller), (except fixed function IOs like Vdd, GND, CCx), can be configured to select one from many available options. This enumerated type lists the various IO functions that can be selected for the flexible IOs. Not all functionality can be configured to any selected IO. The selectable functionality for each IO is fixed and cannot be altered. For example, in case of CCG3 or DMC, SWD\_CLK cannot be configured to any other pin other than P2.1. But P2.1 can be configured to function as GPIO, SCB1\_SPI\_MOSI, SCB1\_I2C\_SDA or SCB1\_UART\_RTS.

Please refer to the respective device datasheets for more information about these IO options.

## Enumerator

|                 |                                           |
|-----------------|-------------------------------------------|
| HSIOM_MODE_GPIO | Special functions disabled. Used as GPIO. |
|-----------------|-------------------------------------------|

### 6.32.3 Function Documentation

#### 6.32.3.1 gpio\_clear\_intr()

```
void gpio_clear_intr (
 gpio_port_pin_t port_pin)
```

Clear interrupt status on the specified GPIO.

This function clears any active interrupts on the specified GPIO.

##### Parameters

|                 |                    |
|-----------------|--------------------|
| <i>port_pin</i> | Pin to be updated. |
|-----------------|--------------------|

##### Returns

None

#### 6.32.3.2 gpio\_get\_intr()

```
bool gpio_get_intr (
 gpio_port_pin_t port_pin)
```

Read the interrupt status on a specific GPIO.

This function checks whether there are any active interrupts on the specified GPIO.

##### Parameters

|                 |                    |
|-----------------|--------------------|
| <i>port_pin</i> | Pin to be queried. |
|-----------------|--------------------|

##### Returns

true if interrupt is active, false otherwise.

#### 6.32.3.3 gpio\_hsiom\_set\_config()

```
void gpio_hsiom_set_config (
 gpio_port_pin_t port_pin,
 hsiom_mode_t hsiom_mode,
 gpio_dm_t drv_mode,
 bool value)
```

Single function for complete configuration of a CCG IO.

This is a single API which can be used to configure the IO mode, the drive mode and the current value of a CCG device IO. No error checks are performed, and the caller should ensure that the settings provided are valid for the selected IO.

**Parameters**

|                   |                       |
|-------------------|-----------------------|
| <i>port_pin</i>   | Pin to be configured. |
| <i>hsiom_mode</i> | Desired IO mode.      |
| <i>drv_mode</i>   | Desired drive mode.   |
| <i>value</i>      | Desired output state. |

**Returns**

None

**6.32.3.4 gpio\_int\_set\_config()**

```
void gpio_int_set_config (
 gpio_port_pin_t port_pin,
 uint8_t int_mode)
```

Configure the GPIO with the desired interrupt setting.

This API configures the interrupt mode for the specified GPIO. It does not do any error check and will just configure the GPIO interrupt setting. Care should be taken to make sure that wrong indexing is not done.

**Parameters**

|                 |                         |
|-----------------|-------------------------|
| <i>port_pin</i> | Pin to be configured.   |
| <i>int_mode</i> | Desired interrupt mode. |

**Returns**

None

**6.32.3.5 gpio\_read\_value()**

```
bool gpio_read_value (
 gpio_port_pin_t port_pin)
```

Get the GPIO current state.

This API retrieves the current state of a pin, assuming it has been configured as a GPIO. It is the caller's responsibility to ensure that the HSIOM and drive mode settings for the pin have been set correctly.

The API does not do any error check and will just read the GPIO state register. Care should be taken to make sure that wrong indexing is not done.

**Parameters**

|                 |                    |
|-----------------|--------------------|
| <i>port_pin</i> | Pin to be queried. |
|-----------------|--------------------|

**Returns**

Current state of the pin.

**See also**

[hsiom\\_set\\_config](#)  
[gpio\\_set\\_drv\\_mode](#)

### 6.32.3.6 gpio\_set\_drv\_mode()

```
void gpio_set_drv_mode (
 gpio_port_pin_t port_pin,
 gpio_dm_t drv_mode)
```

Configure the GPIO with the desired drive mode.

This API updates the drive mode for the selected CCG device IO. The API does not do any error check and will just set the drive mode of GPIO. The caller should ensure that the HSIOM setting for the pin has been selected correctly.

**Parameters**

|                 |                       |
|-----------------|-----------------------|
| <i>port_pin</i> | Pin to be configured. |
| <i>drv_mode</i> | Desired drive mode.   |

**Returns**

None

### 6.32.3.7 gpio\_set\_lvttl\_mode()

```
void gpio_set_lvttl_mode (
 uint8_t port)
```

Set the input buffer voltage for a port to LVTTL.

This API sets the input buffer voltage for a complete CCG device port to LVTTL levels. This is required to be set for ports that include the IOs used for the I2C based HPI interface.

**Parameters**

|             |                                            |
|-------------|--------------------------------------------|
| <i>port</i> | IO port to be configured for LVTTL levels. |
|-------------|--------------------------------------------|

**Returns**

None

### 6.32.3.8 gpio\_set\_value()

```
void gpio_set_value (
 gpio_port_pin_t port_pin,
 bool value)
```

Sets the GPIO to the required state.

This function updates the output state of a GPIO pin. The API does not do any error check and will just update the output state. It is the caller's responsibility to ensure that the HSIOM and drive mode settings for the pin have been selected as required.

Care should be taken to make sure that wrong indexing is not done.

#### Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>port_pin</i> | Pin to be updated.                        |
| <i>value</i>    | Value to drive on the pin: 0=LOW, 1=HIGH. |

#### Returns

None

#### See also

[hsiom\\_set\\_config](#)  
[gpio\\_set\\_drv\\_mode](#)

### 6.32.3.9 hsiom\_set\_config()

```
void hsiom_set_config (
 gpio_port_pin_t port_pin,
 hsiom_mode_t hsiom_mode)
```

Select the IO mode for a CCG device IO.

This API configures the IO mode for a CCG device IO. It does not do any error check and will just set the HSIOM configuration for the GPIO. Care should be taken to make sure that wrong indexing is not done.

#### Parameters

|                   |                       |
|-------------------|-----------------------|
| <i>port_pin</i>   | Pin to be configured. |
| <i>hsiom_mode</i> | Desired IO mode.      |

#### Returns

None

#### See also

[hsiom\\_mode\\_t](#)

## 6.33 system/status.h File Reference

### Macros

- #define [CCG\\_STATUS\\_CODE\\_OFFSET](#) (2)
- #define [CCG\\_STATUS\\_TO\\_HPI\\_RESPONSE](#)(c) ((c) + [CCG\\_STATUS\\_CODE\\_OFFSET](#))

## Enumerations

- enum `ccg_status_t` {
 `CCG_STAT_NO_RESPONSE` = -2, `CCG_STAT_SUCCESS` = 0, `CCG_STAT_FLASH_DATA_AVAILABLE`,  
`CCG_STAT_BAD_PARAM`,  
`CCG_STAT_INVALID_COMMAND` = 3, `CCG_STAT_FLASH_UPDATE_FAILED` = 5, `CCG_STAT_INVALID_FW`,  
`CCG_STAT_INVALID_ARGUMENT`,  
`CCG_STAT_NOT_SUPPORTED`, `CCG_STAT_INVALID_SIGNATURE`, `CCG_STAT_TRANS_FAILURE`,  
`CCG_STAT_CMD_FAILURE`,  
`CCG_STAT_FAILURE`, `CCG_STAT_READ_DATA`, `CCG_STAT_NOT_READY`, `CCG_STAT_BUSY`,  
`CCG_STAT_TIMEOUT` }

### 6.33.1 Detailed Description

API return status definitions.

### 6.33.2 Macro Definition Documentation

#### 6.33.2.1 CCG\_STATUS\_CODE\_OFFSET

```
#define CCG_STATUS_CODE_OFFSET (2)
```

Value to be added to the status code to get the HPI/UVDM response code.

The HPI/UVDM interface response codes use a different convention than the core function return codes. This value represents the offset that should be added to the function return code to get the HPI/UVDM response code.

### 6.33.3 Enumeration Type Documentation

#### 6.33.3.1 ccg\_status\_t

```
enum ccg_status_t
```

Interface status codes.

Enumeration to hold status codes for all CCG interfaces. These values are pre-defined for each interface and should not be modified. To make interface usage easier, the enumeration starts at -2. This allows the success status to have a value of zero. The response code should be incremented by two before sending out on the individual interfaces.

#### Enumerator

|                                            |                                                         |
|--------------------------------------------|---------------------------------------------------------|
| <code>CCG_STAT_NO_RESPONSE</code>          | Special status code indicating no response.             |
| <code>CCG_STAT_SUCCESS</code>              | Success status.                                         |
| <code>CCG_STAT_FLASH_DATA_AVAILABLE</code> | Special status code indicating flash data availability. |
| <code>CCG_STAT_BAD_PARAM</code>            | Bad input parameter.                                    |
| <code>CCG_STAT_INVALID_COMMAND</code>      | Operation failed due to invalid command.                |
| <code>CCG_STAT_FLASH_UPDATE_FAILED</code>  | Flash write operation failed.                           |
| <code>CCG_STAT_INVALID_FW</code>           | Special status code indicating invalid firmware         |
| <code>CCG_STAT_INVALID_ARGUMENT</code>     | Operation failed due to invalid arguments.              |
| <code>CCG_STAT_NOT_SUPPORTED</code>        | Feature not supported.                                  |

**Enumerator**

|                            |                                                        |
|----------------------------|--------------------------------------------------------|
| CCG_STAT_INVALID_SIGNATURE | Invalid signature parameter identified.                |
| CCG_STAT_TRANS_FAILURE     | Transaction failure status.                            |
| CCG_STAT_CMD_FAILURE       | Command failure status                                 |
| CCG_STAT_FAILURE           | Generic failure status.                                |
| CCG_STAT_READ_DATA         | Special status code indicating read data availability. |
| CCG_STAT_NOT_READY         | Operation failed due to device/stack not ready.        |
| CCG_STAT_BUSY              | Operation failed due to device/stack busy status.      |
| CCG_STAT_TIMEOUT           | Operation timed out.                                   |

**6.34 system/system.h File Reference**

```
#include "stdint.h"
```

**Macros**

- #define **SYS\_BOOT\_VERSION\_ADDRESS** (0x000000E0)
- #define **SYS\_FW\_VERSION\_OFFSET** (0x000000E0)
- #define **SYS\_APP\_VERSION\_OFFSET** (0x00000004)
- #define **SYS\_SILICON\_ID\_OFFSET** (0x000000EA)
- #define **SYS\_BOOT\_TYPE\_FIELD\_OFFSET** (0x000000EC)
- #define **SYS\_FW\_CUSTOM\_INFO\_OFFSET** (0x000000C0)
- #define **SYS\_METADATA\_VALID\_SIG** (0x4359)
- #define **SYS\_PSEUDO\_METADATA\_VALID\_SIG** (0x4350)
- #define **SYS\_BOOT\_MODE\_RQT\_SIG** (0x424C)
- #define **SYS\_CONFIG\_TABLE\_SIGN** (0x4359u)
- #define **SYS\_INVALID\_FW\_START\_ADDR** (0x00000000)
- #define **SYS\_BOOT\_TYPE\_APP\_PRIORITY\_POS** (0x02)
- #define **SYS\_BOOT\_TYPE\_FW\_UPDATE\_INTERFACE\_POS** (0x01)
- #define **SYS\_BOOT\_TYPE\_SECURE\_BOOT\_MASK** (0x01)
- #define **SYS\_BOOT\_TYPE\_FW\_UPDATE\_INTERFACE\_MASK** (0x02)

**Enumerations**

- enum **sys\_fw\_mode\_t**{ **SYS\_FW\_MODE\_BOOTLOADER** = 0, **SYS\_FW\_MODE\_FWIMAGE\_1**, **SYS\_FW\_MODE\_FWIMAGE\_2**, **SYS\_FW\_MODE\_INVALID** }

**Functions**

- void **sys\_set\_device\_mode** (**sys\_fw\_mode\_t** fw\_mode)
- **sys\_fw\_mode\_t sys\_get\_device\_mode** (void)
- **uint8\_t \* sys\_get\_boot\_version** (void)
- **uint8\_t \* sys\_get\_img1\_fw\_version** (void)
- **uint8\_t \* sys\_get\_img2\_fw\_version** (void)
- **uint32\_t sys\_get\_fw\_img1\_start\_addr** (void)
- **uint32\_t sys\_get\_fw\_img2\_start\_addr** (void)
- **uint8\_t sys\_get\_recent\_fw\_image** (void)
- void **sys\_get\_silicon\_id** (**uint32\_t** \*silicon\_id)

- `uint8_t get_silicon_revision (void)`
- `uint32_t sys_get_custom_info_addr (void)`
- `uint16_t sys_get_bcdDevice_version (uint32_t ver_addr)`

## Variables

- `sys_fw_mode_t gl_active_fw`
- `uint8_t gl_invalid_version [8]`

### 6.34.1 Detailed Description

Support functions and definitions for bootloader and flash updates.

### 6.34.2 Enumeration Type Documentation

#### 6.34.2.1 `sys_fw_mode_t`

```
enum sys_fw_mode_t
```

List of CCG firmware modes.

#### Enumerator

|                                     |                   |
|-------------------------------------|-------------------|
| <code>SYS_FW_MODE_BOOTLOADER</code> | Bootloader mode.  |
| <code>SYS_FW_MODE_FWIMAGE_1</code>  | Firmware Image #1 |
| <code>SYS_FW_MODE_FWIMAGE_2</code>  | Firmware Image #2 |
| <code>SYS_FW_MODE_INVALID</code>    | Invalid value.    |

### 6.34.3 Function Documentation

#### 6.34.3.1 `get_silicon_revision()`

```
uint8_t get_silicon_revision (
 void)
```

Returns Silicon revision.

#### Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <code>None</code> | <input type="button" value=""/> |
|-------------------|---------------------------------|

#### Returns

Silicon revision B[7:4] - Major rev B[3:0] - Minor rev

### 6.34.3.2 sys\_get\_bcdDevice\_version()

```
uint16_t sys_get_bcdDevice_version (
 uint32_t ver_addr)
```

Get bcdDevice version of device.

This function returns bcdDevice version for the device which can be used as part of D\_ID response, secure boot checks etc. Format of bcdDevice version is documented in the function body.

#### Parameters

|                       |                                    |
|-----------------------|------------------------------------|
| <code>ver_addr</code> | Offset of version in Flash memory. |
|-----------------------|------------------------------------|

#### Returns

16 bits bcdDevice version.

### 6.34.3.3 sys\_get\_boot\_version()

```
uint8_t* sys_get_boot_version (
 void)
```

Get bootloader version.

The bootloader version is stored at absolute address SYS\_CCG\_BOOT\_VERSION\_ADDRESS in device FLASH. This function returns a pointer to this version information.

#### Returns

Pointer to the bootloader version information.

### 6.34.3.4 sys\_get\_custom\_info\_addr()

```
uint32_t sys_get_custom_info_addr (
 void)
```

Get start address of Customer info section.

This function returns the start address of Customer info section.

#### Parameters

|                   |                          |
|-------------------|--------------------------|
| <code>None</code> | <input type="checkbox"/> |
|-------------------|--------------------------|

#### Returns

Address of Customer info section.

### 6.34.3.5 sys\_get\_device\_mode()

```
sys_fw_mode_t sys_get_device_mode (
```

```
 void)
```

Get the current firmware mode.

This function retrieves the current firmware mode of the CCG device.

#### Returns

The current firmware mode.

#### 6.34.3.6 sys\_get\_fw\_img1\_start\_addr()

```
uint32_t sys_get_fw_img1_start_addr (
 void)
```

Get the flash start address of firmware image-1.

This function returns the flash address from where firmware image-1 (FW1) has been stored.

#### Returns

Start address of firmware image-1.

#### 6.34.3.7 sys\_get\_fw\_img2\_start\_addr()

```
uint32_t sys_get_fw_img2_start_addr (
 void)
```

Get the flash start address of firmware image-2.

This function returns the flash address from where firmware image-2 (FW2) has been stored.

#### Returns

Start address of firmware image-2.

#### 6.34.3.8 sys\_get\_img1\_fw\_version()

```
uint8_t* sys_get_img1_fw_version (
 void)
```

Get version for firmware image-1.

This function returns a pointer to the version information for firmware image-1 (FW1). The version is located at a fixed offset of CY\_PD\_FW\_VERSION\_OFFSET bytes from the start of the firmware binary.

#### Returns

Pointer to the firmware image-1 version information.

### 6.34.3.9 sys\_get\_img2\_fw\_version()

```
uint8_t* sys_get_img2_fw_version (
 void)
```

Get version for firmware image-2.

This function returns a pointer to the version information for firmware image-2 (FW2). The version is located at a fixed offset of CY\_PD\_FW\_VERSION\_OFFSET bytes from the start of the firmware binary.

#### Returns

Pointer to the firmware image-2 version information.

### 6.34.3.10 sys\_get\_recent\_fw\_image()

```
uint8_t sys_get_recent_fw_image (
 void)
```

Determines the more recently update firmware image.

The CCG Bootloader uses this function to determine the more recently updated firmware image (from among F $\leftarrow$  W1 and FW2) by comparing the sequence numbers of images which are stored in the firmware metadata table. The bootloader loads the most recently updated binary by default (even if its version is older than that of the other firmware binary).

#### Returns

Firmware id: 1 for Image-1 and 2 for Image-2.

### 6.34.3.11 sys\_get\_silicon\_id()

```
void sys_get_silicon_id (
 uint32_t * silicon_id)
```

Get Silicon ID of device.

This function retrieves the Silicon ID of the CCG device.

#### Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>silicon_id</i> | Pointer to buffer to hold the Silicon ID. |
|-------------------|-------------------------------------------|

#### Returns

None

### 6.34.3.12 sys\_set\_device\_mode()

```
void sys_set_device_mode (
 sys_fw_mode_t fw_mode)
```

Set the current firmware mode.

This function is used by the start-up logic to store the current firmware mode for the CCGx device.

This should not be used outside of the default start-up logic for the CCGx bootloader and firmware applications.

#### Parameters

|                      |                                     |
|----------------------|-------------------------------------|
| <code>fw_mode</code> | The active firmware mode to be set. |
|----------------------|-------------------------------------|

#### Returns

None

## 6.35 system/timer.h File Reference

```
#include "config.h"
#include "stdbool.h"
```

### Data Structures

- struct `ccg_timer_t`

### Macros

- #define `TIMER_NUM_TIMERS` (31u)
- #define `TIMER_MAX_TIMEOUT` (0xFFFF)
- #define `TIMER_INVALID_ID` (0xFFu)
- #define `TIMER_INVALID_INDEX` (0xFFu)

### TypeDefs

- typedef uint8\_t `timer_id_t`
- typedef void(\* `timer_cb_t`) (uint8\_t instance, `timer_id_t` id)

### Functions

- void `timer_init` (void)
- bool `timer_start` (uint8\_t instance, `timer_id_t` id, uint16\_t period, `timer_cb_t` cb)
- bool `timer_start_woCb` (uint8\_t instance, `timer_id_t` id, uint16\_t period)
- void `timer_stop` (uint8\_t instance, `timer_id_t` id)
- bool `timer_is_running` (uint8\_t instance, `timer_id_t` id)
- uint16\_t `timer_get_count` (uint8\_t instance, `timer_id_t` id)
- void `timer_stop_all` (uint8\_t instance)
- void `timer_stop_range` (uint8\_t instance, `timer_id_t` start, `timer_id_t` stop)
- uint8\_t `timer_num_active` (uint8\_t instance)
- void `timer_enter_sleep` (void)
- uint16\_t `timer_get_multiplier` (void)

### 6.35.1 Detailed Description

Soft timer header file.

## 6.35.2 Typedef Documentation

### 6.35.2.1 timer\_cb\_t

`timer_cb_t`

Timer callback function.

This callback function is invoked on timer expiry and should be treated as interrupt.

### 6.35.2.2 timer\_id\_t

`timer_id_t`

Timer ID type definition.

This type definition is used to identify software timer objects. The timer ID needs to be unique for each soft timer instance and need to be maintained in the application. To maintain uniqueness, the following timer ID allocation rule is expected to be followed.

PD and Type-C stack ([pd.h](#)) : 0 - 29 : 30 timers Base application stack ([app.h](#)) : 30 - 127 : 98 timers HPI module ([hpi.h](#)) : 128 - 135 : 8 timers I2C module ([i2c.h](#)) : 136 - 143 : 8 timers USB module ([usb.h](#)) : 144 - 151 : 8 timers IECS module ([iecs.h](#)) : 152 - 159 : 8 timers Reserved : 160 - 191 : 32 timers Solution (project directory) : 192 - 254 : 63 timers

## 6.35.3 Function Documentation

### 6.35.3.1 timer\_enter\_sleep()

```
void timer_enter_sleep (
 void)
```

Prepare the timer hardware for entering deep sleep.

This function prepares the timer module and the hardware timer for entering device deep sleep. This must be called prior to entering deep sleep mode.

#### Returns

None

### 6.35.3.2 timer\_get\_count()

```
uint16_t timer_get_count (
 uint8_t instance,
 timer_id_t id)
```

Returns the time remaining for timer expiration.

Returns the time remaining for timer expiration.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>instance</code> | Instance number for which we are using the timer. |
|-----------------------|---------------------------------------------------|

**Parameters**

|           |                  |
|-----------|------------------|
| <i>id</i> | Unique timer id. |
|-----------|------------------|

**Returns**

Time remaining for expiration of the soft timer.

**6.35.3.3 timer\_get\_multiplier()**

```
uint16_t timer_get_multiplier (
 void)
```

Get the number of LF clock ticks required per ms.

**Returns**

Returns the number of LF clock ticks per ms.

**6.35.3.4 timer\_init()**

```
void timer_init (
 void)
```

Initialize the software timer module.

This function initializes the software timer module. This function initializes the data structures for timer management and enables the hardware timer used for the soft timer implementation.

**Returns**

None

**6.35.3.5 timer\_is\_running()**

```
bool timer_is_running (
 uint8_t instance,
 timer_id_t id)
```

Check whether the specified soft timer is currently running.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>instance</i> | Instance number for which we are using the timer. |
| <i>id</i>       | Unique timer id.                                  |

**Returns**

true if the timer is running, false otherwise.

### 6.35.3.6 timer\_num\_active()

```
uint8_t timer_num_active (
 uint8_t instance)
```

Returns number of active timers.

#### Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | Instance number to query. |
|-----------------|---------------------------|

#### Returns

Number of active timers on this instance.

### 6.35.3.7 timer\_start()

```
bool timer_start (
 uint8_t instance,
 timer_id_t id,
 uint16_t period,
 timer_cb_t cb)
```

Start a soft timer.

Start a specific soft timer. All soft timers are one-shot timers which will run until the specified period has elapsed. The timer expiration callback will be called at the end of the period, if one is provided.

#### Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>instance</i> | Timer instance number for which we are using the timer. |
| <i>id</i>       | Unique timer id                                         |
| <i>period</i>   | Timer period in milliseconds.                           |
| <i>cb</i>       | Timer expiration callback. Can be NULL.                 |

#### Returns

true if the timer is started, false if timer start fails.

### 6.35.3.8 timer\_start\_wocb()

```
bool timer_start_wocb (
 uint8_t instance,
 timer_id_t id,
 uint16_t period)
```

Start a soft timer.

Start a specific soft timer. All soft timers are one-shot timers which will run until the specified period has elapsed.

#### Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>instance</i> | Timer instance number for which we are using the timer. |
|-----------------|---------------------------------------------------------|

**Parameters**

|               |                               |
|---------------|-------------------------------|
| <i>id</i>     | Unique timer id               |
| <i>period</i> | Timer period in milliseconds. |

**Returns**

true if the timer is started, false if timer start fails.

**6.35.3.9 timer\_stop()**

```
void timer_stop (
 uint8_t instance,
 timer_id_t id)
```

Stop a soft timer.

Stop a soft timer which is currently running.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>instance</i> | Instance number for which we are using the timer. |
| <i>id</i>       | Unique timer id.                                  |

**Returns**

None

**6.35.3.10 timer\_stop\_all()**

```
void timer_stop_all (
 uint8_t instance)
```

Stops all soft timers associated with the instance.

**Parameters**

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | Instance number on which timers are to be stopped. |
|-----------------|----------------------------------------------------|

**Returns**

None

**6.35.3.11 timer\_stop\_range()**

```
void timer_stop_range (
 uint8_t instance,
 timer_id_t start,
 timer_id_t stop)
```

This function stops all soft timers with ids in the specified range.

#### Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>instance</i> | Instance number on which soft timers are to be stopped. |
| <i>start</i>    | Starting timer ID. The value is inclusive.              |
| <i>stop</i>     | Ending timer ID. The value is inclusive.                |

#### Returns

None

## 6.36 system/utils.h File Reference

```
#include "stdint.h"
```

#### Macros

- #define **GET\_MAX**(a, b) (((a) > (b)) ? (a) : (b))
- #define **GET\_MIN**(a, b) (((a) > (b)) ? (b) : (a))
- #define **DIV\_ROUND\_UP**(x, y) (((x) + ((y) - 1)) / (y))
- #define **DIV\_ROUND\_NEAREST**(x, y) (((x) + ((y) / 2)) / (y))
- #define **MAKE\_WORD**(hi, lo) (((uint16\_t)(hi) << 8) | ((uint16\_t)(lo)))
- #define **MAKE\_DWORD**(b3, b2, b1, b0)
- #define **MAKE\_DWORD\_FROM\_WORD**(hi, lo) (((uint32\_t)(hi) << 16) | ((uint32\_t)(lo)))
- #define **WORD\_GET\_MSB**(w) ((uint8\_t)((w) >> 8))
- #define **WORD\_GET\_LSB**(w) ((uint8\_t)((w) & 0xFF))
- #define **BYTE\_GET\_UPPER\_NIBBLE**(w) ((uint8\_t)((w) >> 4) & 0xF)
- #define **BYTE\_GET\_LOWER\_NIBBLE**(w) ((uint8\_t)((w) & 0xF))
- #define **DWORD\_GET\_BYTE0**(dw) ((uint8\_t)((dw) & 0xFF))
- #define **DWORD\_GET\_BYTE1**(dw) ((uint8\_t)((dw) >> 8) & 0xFF))
- #define **DWORD\_GET\_BYTE2**(dw) ((uint8\_t)((dw) >> 16) & 0xFF))
- #define **DWORD\_GET\_BYTE3**(dw) ((uint8\_t)((dw) >> 24) & 0xFF))
- #define **MEM\_COPY**(dest, src, size) memcpy ((uint8\_t \*)(dest), (uint8\_t \*)(src), (size))
- #define **MEM\_CMP**(buf1, buf2, size) memcmp ((uint8\_t \*)(buf1), (uint8\_t \*)(buf2), (size))
- #define **MEM\_SET**(dest, byte, size) memset ((uint8\_t \*)(dest), (byte), (size))
- #define **BUSY\_WAIT\_US**(us) CyDelayCycles ((uint32\_t)(us))
- #define **REV\_BYTE\_ORDER**(n, b, i)

#### Functions

- uint8\_t **mem\_calculate\_byte\_checksum** (uint8\_t \*ptr, uint32\_t size)
- uint16\_t **mem\_calculate\_word\_checksum** (uint16\_t \*ptr, uint32\_t size)
- uint32\_t **mem\_calculate\_dword\_checksum** (uint32\_t \*ptr, uint32\_t size)
- uint16\_t **crc16** (uint16\_t crc, uint8\_t data)
- void **mem\_copy\_word** (uint32\_t \*dest, const uint32\_t \*source, uint32\_t size)

### 6.36.1 Detailed Description

General utility macros and definitions for CCGx firmware stack.

## 6.36.2 Macro Definition Documentation

### 6.36.2.1 MAKE\_DWORD

```
#define MAKE_DWORD (
 b3,
 b2,
 b1,
 b0)
```

**Value:**

```
(((uint32_t) (b3) << 24) | ((uint32_t) (b2) << 16) | \
 ((uint32_t) (b1) << 8) | ((uint32_t) (b0)))
```

Combine four bytes to create one 32-bit DWORD.

### 6.36.2.2 REV\_BYTE\_ORDER

```
#define REV_BYTE_ORDER (
 n,
 b,
 i)
```

**Value:**

```
(n) = ((uint32_t) (b)[(i)] << 24) \
| ((uint32_t) (b)[(i) + 1] << 16) \
| ((uint32_t) (b)[(i) + 2] << 8) \
| ((uint32_t) (b)[(i) + 3]) ; \
```

Reverse byte order in a 32-Bit Integer. n - Final 32 but number b - Source i - Index into Source.

## 6.36.3 Function Documentation

### 6.36.3.1 crc16()

```
uint16_t crc16 (
 uint16_t crc,
 uint8_t data)
```

Function to calculate 16-bit CRC.

The function implements CRC-16 with polynomial  $x^{16} + x^{15} + x^2 + 1$  (0xA001).

**Parameters**

|             |                                              |
|-------------|----------------------------------------------|
| <i>crc</i>  | Original CRC value.                          |
| <i>data</i> | Data byte to be included in CRC computation. |

**Returns**

Updated CRC value including the new data byte.

**6.36.3.2 mem\_calculate\_byte\_checksum()**

```
uint8_t mem_calculate_byte_checksum (
 uint8_t * ptr,
 uint32_t size)
```

Calculate the 2's complement binary checksum over a BYTE array.

This function calculates the checksum of the specified byte array. The checksum is a simple function calculated as the 2's complement of the binary sum of all bytes in the array. This checksum is used for the firmware binary as well as the configuration table.

**Parameters**

|             |                                     |
|-------------|-------------------------------------|
| <i>ptr</i>  | Pointer to the data array.          |
| <i>size</i> | Size of the array in BYTE elements. |

**Returns**

Checksum of the data array.

**6.36.3.3 mem\_calculate\_dword\_checksum()**

```
uint32_t mem_calculate_dword_checksum (
 uint32_t * ptr,
 uint32_t size)
```

Calculate the 2's complement binary checksum over a DWORD array.

This function calculates the checksum of the specified DWORD array. The checksum is a simple function calculated as the 2's complement of the binary sum of all d-words in the array. This checksum is used for the firmware binary as well as the configuration table.

**Parameters**

|             |                                      |
|-------------|--------------------------------------|
| <i>ptr</i>  | Pointer to the data array.           |
| <i>size</i> | Size of the array in DWORD elements. |

**Returns**

Checksum of the data array.

**6.36.3.4 mem\_calculate\_word\_checksum()**

```
uint16_t mem_calculate_word_checksum (
 uint16_t * ptr,
 uint32_t size)
```

Calculate the 2's complement binary checksum over a WORD array.

This function calculates the checksum of the specified WORD array. The checksum is a simple function calculated as the 2's complement of the binary sum of all words in the array.

#### Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>ptr</i>  | Pointer to the data array.          |
| <i>size</i> | Size of the array in WORD elements. |

#### Returns

Checksum of the data array.

#### 6.36.3.5 mem\_copy\_word()

```
void mem_copy_word (
 uint32_t * dest,
 const uint32_t * source,
 uint32_t size)
```

Function to copy 32bit data from one location to another.

#### Parameters

|             |                         |
|-------------|-------------------------|
| <i>dest</i> | Pointer to Destination. |
| <i>data</i> | Pointer to source.      |
| <i>size</i> | Size of data.           |

#### Returns

None.

## 6.37 usb/usb.h File Reference

```
#include "stdint.h"
#include "stdbool.h"
#include "config.h"
#include "status.h"
#include "usbconst.h"
```

### Data Structures

- struct [usb\\_config\\_t](#)
- struct [usb\\_ep\\_handle\\_t](#)
- struct [usb\\_handle\\_t](#)
- struct [USBSIE\\_REGS\\_T](#)
- struct [USBARB\\_REGS\\_T](#)
- struct [USBARB16\\_REGS\\_T](#)

## Macros

- #define **USB\_NUM\_EP** (8)
- #define **USB\_EP\_MAX\_PKT\_SIZE** (64)
- #define **USB\_BUF\_SIZE\_64BYTE** (0x66)
- #define **USB\_EP\_HW\_BUFFER\_SIZE** (0x200)
- #define **USB\_EPO\_SIZE** (8)
- #define **USB\_NUM\_CONFIGURATION** (1)
- #define **USB\_NUM\_ALT\_INTERFACE** (1)
- #define **USB\_NUM\_INTERFACE** (4)
- #define **USB\_TIMER\_INSTANCE** (0)
- #define **USB\_TIMER\_ID\_BASE** (144u)
- #define **USB\_SUSPEND\_TIMER** (**USB\_TIMER\_ID\_BASE**)
- #define **USB\_SUSPEND\_TIMER\_PERIOD** (10)
- #define **USB\_REMOTE\_WAKEUP\_TIMER** (145u)
- #define **USB\_REMOTE\_WAKEUP\_TIMER\_PERIOD** (15)
- #define **USB\_VBUS\_REG\_EN\_THRESHOLD** (3700)
- #define **USBDEV\_SIE\_EPx\_CNT0\_ADDRESS** (**USBDEV\_SIE\_EP1\_CNT0\_ADDRESS**)
- #define **USBDEV\_SIE\_EPx\_CNT0**(m) (\*(volatile uint32\_t \*)(**USBDEV\_SIE\_EP1\_CNT0\_ADDRESS** + ((m) \* 0x40)))
- #define **USBDEV\_SIE\_EPx\_CNT0\_DEFAULT** (0x00000000)
- #define **USBDEV\_SIE\_EPx\_CNT0\_DATA\_COUNT\_MSB\_MASK** (0x00000007) /\* <0:2> RW:RW:0: \*/
- #define **USBDEV\_SIE\_EPx\_CNT0\_DATA\_COUNT\_MSB\_POS** (0)
- #define **USBDEV\_SIE\_EPx\_CNT0\_DATA\_VALID** (1u << 6) /\* <6:6> RW1S:RW0C:0: \*/
- #define **USBDEV\_SIE\_EPx\_CNT0\_DATA\_TOGGLE** (1u << 7) /\* <7:7> RW:RW:0: \*/
- #define **USBDEV\_SIE\_EPx\_CNT1\_ADDRESS** (**USBDEV\_SIE\_EP1\_CNT1\_ADDRESS**)
- #define **USBDEV\_SIE\_EPx\_CNT1**(m) (\*(volatile uint32\_t \*)(**USBDEV\_SIE\_EP1\_CNT1\_ADDRESS** + ((m) \* 0x40)))
- #define **USBDEV\_SIE\_EPx\_CNT1\_DEFAULT** (0x00000000)
- #define **USBDEV\_SIE\_EPx\_CNT1\_DATA\_COUNT\_MASK** (0x000000ff) /\* <0:7> RW:RW:0: \*/
- #define **USBDEV\_SIE\_EPx\_CNT1\_DATA\_COUNT\_POS** (0)
- #define **USBDEV\_SIE\_EPx\_CR0\_ADDRESS** (**USBDEV\_SIE\_EPx\_CR0\_ADDRESS**)
- #define **USBDEV\_SIE\_EPx\_CR0**(m) (\*(volatile uint32\_t \*)(**USBDEV\_SIE\_EPx\_CR0\_ADDRESS**))
- #define **USBDEV\_SIE\_EPx\_CR0\_DEFAULT** (0x00000000)
- #define **USBDEV\_SIE\_EPx\_CR0\_MODE\_MASK** (0x0000000f) /\* <0:3> RW:RW:0: \*/
- #define **USBDEV\_SIE\_EPx\_CR0\_MODE\_POS** (0)
- #define **USBDEV\_SIE\_EPx\_CR0\_ACKED\_TXN** (1u << 4) /\* <4:4> RW1S:RWC:0: \*/
- #define **USBDEV\_SIE\_EPx\_CR0\_NAK\_INT\_EN** (1u << 5) /\* <5:5> R:RW:0: \*/
- #define **USBDEV\_SIE\_EPx\_CR0\_ERR\_IN\_TXN** (1u << 6) /\* <6:6> RW1S:RWC:0: \*/
- #define **USBDEV\_SIE\_EPx\_CR0\_STALL** (1u << 7) /\* <7:7> R:RW:0: \*/
- #define **USBDEV\_ARB\_EPx\_CFG\_IN\_DATA\_RDY** (1u << 0) /\* <0:0> R:RW:0: \*/
- #define **USBDEV\_ARB\_EPx\_CFG\_DMA\_REQ** (1u << 1) /\* <1:1> R:RW:0: \*/
- #define **USBDEV\_ARB\_EPx\_CFG\_CRC\_BYPASS** (1u << 2) /\* <2:2> R:RW:0: \*/
- #define **USBDEV\_ARB\_EPx\_CFG\_RESET\_PTR** (1u << 3) /\* <3:3> R:RW:0: \*/
- #define **USBDEV\_ARB\_RWx\_DR\_ADDRESS**(m) (**USBDEV\_ARB\_RW1\_DR\_ADDRESS** + ((m) \* (0x40)))
- #define **USBDEV\_ARB\_RWx\_DR**(m) (\*(volatile uint32\_t \*)(**USBDEV\_ARB\_RW1\_DR\_ADDRESS** + ((m) \* 0x40)))
- #define **USBDEV\_ARB\_RWx\_DR\_DEFAULT** (0x00000000)
- #define **USBDEV\_ARB\_RWx\_DR16\_ADDRESS**(m) (**USBDEV\_ARB\_RW1\_DR16\_ADDRESS** + ((m) \* (0x40)))
- #define **USBDEV\_ARB\_RWx\_DR16**(m) (\*(volatile uint32\_t \*)(**USBDEV\_ARB\_RW1\_DR16\_ADDRESS** + ((m) \* 0x40)))
- #define **USBDEV\_ARB\_RWx\_DR16\_DEFAULT** (0x00000000)

## Typedefs

- `typedef ccg_status_t(* usb_setup_cb_t) (usb_setup_pkt_t *setup_pkt)`
- `typedef void(* usb_event_cb_t) (usb_state_t state, uint32_t data)`
- `typedef struct USBSIE_REGS_T * PUSBSIE_REGS_T`
- `typedef struct USBARB_REGS_T * PUSBARB_REGS_T`
- `typedef struct USBARB16_REGS_T * PUSBARB16_REGS_T`

## Enumerations

- `enum usb_state_t {`  
`USB_STATE_DEINITED = 0, USB_STATE_DISABLED, USB_STATE_WAIT_FOR_VBUS, USB_STATE_CONNECTED,`  
`USB_STATE_RESET, USB_STATE_ADDRESSED, USB_STATE_CONFIGURED, USB_STATE_SUSPENDED,`  
`USB_STATE_RESUMED, USB_STATE_UNCONFIGURED }`
- `enum usb_ep0_state_t {`  
`USB_EP0_STATE_DISABLED = 0, USB_EP0_STATE_SETUP, USB_EP0_STATE_DATA_IN, USB_EP0_STATE_DATA_OUT,`  
`USB_EP0_STATE_STATUS_IN, USB_EP0_STATE_STATUS_OUT, USB_EP0_STATE_STALL }`
- `enum usb_ep_index_t {`  
`USB_EP_INDEX_EP1, USB_EP_INDEX_EP2, USB_EP_INDEX_EP3, USB_EP_INDEX_EP4,`  
`USB_EP_INDEX_EP5, USB_EP_INDEX_EP6, USB_EP_INDEX_EP7, USB_EP_INDEX_EP8 }`
- `enum usbdev_ep_mode_t {`  
`USBDEV_EP_MODE_DISABLE, USBDEV_EP_MODE_NAK_INOUT, USBDEV_EP_MODE_STATUS_INOUT_ONLY,`  
`USBDEV_EP_MODE_ISO_OUT = 5, USBDEV_EP_MODE_STATUS_IN_ONLY, USBDEV_EP_MODE_ISO_IN,`  
`USBDEV_EP_MODE_NAK_OUT,`  
`USBDEV_EP_MODE_ACK_OUT, USBDEV_EP_MODE_ACK_OUT_STATUS_IN = 11, USBDEV_EP_MODE_NAK_IN,`  
`USBDEV_EP_MODE_ACK_IN,`  
`USBDEV_EP_MODE_ACK_IN_STATUS_OUT = 15 }`

## Functions

- `void usb_vbus_int_handler (bool vbus_state)`
- `uint32_t usb_intr_lock (void)`
- `void usb_intr_unlock (uint32_t lock)`
- `void usb_task (void)`
- `ccg_status_t usb_init (usb_config_t *cfg)`
- `ccg_status_t usb_enable (bool reg_enable)`
- `ccg_status_t usb_disable (void)`
- `usb_state_t usb_get_state (void)`
- `bool usb_is_idle (void)`
- `ccg_status_t usb_remote_wakeup (void)`
- `ccg_status_t usb_ep0_send_recv_status (void)`
- `ccg_status_t usb_ep0_wait_for_ack (void)`
- `ccg_status_t usb_ep0_set_stall (void)`
- `ccg_status_t usb_ep0_setup_read (uint8_t *data, uint16_t length, bool last, usb_setup_cb_t cb)`
- `ccg_status_t usb_ep0_setup_write (uint8_t *data, uint16_t length, bool last, usb_setup_cb_t cb)`
- `ccg_status_t usb_ep_enable (usb_ep_index_t ep_index, bool is_out)`
- `ccg_status_t usb_ep_disable (usb_ep_index_t ep_index)`
- `ccg_status_t usb_ep_set_stall (usb_ep_index_t ep_index)`
- `ccg_status_t usb_ep_clear_stall (usb_ep_index_t ep_index)`
- `bool usb_ep_is_ready (usb_ep_index_t ep_index)`
- `ccg_status_t usb_ep_queue_read_single (usb_ep_index_t ep_index)`
- `ccg_status_t usb_ep_read_single (usb_ep_index_t ep_index, uint8_t *data, uint8_t *count)`
- `ccg_status_t usb_ep_send_zlp (usb_ep_index_t ep_index)`
- `ccg_status_t usb_ep_write_single (usb_ep_index_t ep_index, uint8_t *data, uint8_t count)`
- `ccg_status_t usb_ep_flush (usb_ep_index_t ep_index)`

## Variables

- const `PUSBSIE_REGS_T USBSIE []`
- const `PUSBARB_REGS_T USBARB []`
- const `PUSBARB16_REGS_T USBARB16 []`

### 6.37.1 Detailed Description

USB driver header file for CCG3 & DMC (Dock Management Controller).

### 6.37.2 Macro Definition Documentation

#### 6.37.2.1 `USB_REMOTE_WAKEUP_TIMER_PERIOD`

```
#define USB_REMOTE_WAKEUP_TIMER_PERIOD (15)
```

USB bus remote wakeup signalling duration in ms.

A remote wakeup is indicated by signalling K-signal on the USB 2.0 bus for 15ms. This period counts the time the signal should be driven.

#### 6.37.2.2 `USB_SUSPEND_TIMER_PERIOD`

```
#define USB_SUSPEND_TIMER_PERIOD (10)
```

USB bus suspend condition detection timeout in ms.

If no SOF is received within 3ms, the USB bus is defined to be in suspend mode. But the timing from USB 2.0 specification is being relaxed here to accomodate for hosts which are not able to send SOFs within 3ms after resume.

#### 6.37.2.3 `USBDEV_ARB_EPx_CFG_CRC_BYPASS`

```
#define USBDEV_ARB_EPx_CFG_CRC_BYPASS (1u << 2) /* <2:2> R:RW:0: */
```

Configuration Setting to prevent CRC bytes from being written to memory and being read by firmware

#### 6.37.2.4 `USBDEV_ARB_EPx_CFG_DMA_REQ`

```
#define USBDEV_ARB_EPx_CFG_DMA_REQ (1u << 1) /* <1:1> R:RW:0: */
```

Manual DMA Request for a particular (1 to 8) endpoint; changing this field from 0 to 1 causes a DMA request to be generated.

#### 6.37.2.5 `USBDEV_ARB_EPx_CFG_IN_DATA_RDY`

```
#define USBDEV_ARB_EPx_CFG_IN_DATA_RDY (1u << 0) /* <0:0> R:RW:0: */
```

Indication that Endpoint Packet Data is Ready in Main memory

### 6.37.2.6 USBDEV\_ARB\_EPx\_CFG\_RESET\_PTR

```
#define USBDEV_ARB_EPx_CFG_RESET_PTR (1u << 3) /* <3:3> R:RW:0: */
```

Configuration Setting to Reset the RA and WA Pointers to their start values at the End of Packet transaction.

### 6.37.2.7 USBDEV\_ARB\_RWx\_DR16\_ADDRESS

```
#define USBDEV_ARB_RWx_DR16_ADDRESS (m) (USBDEV_ARB_RW1_DR16_ADDRESS + ((m) * (0x40)))
```

Endpoint Data Register Data Register for Endpoint. Note: ARB\_RWx\_DR16 and ARB\_RWx\_DR cannot both be used to access the same data packet.

### 6.37.2.8 USBDEV\_ARB\_RWx\_DR\_ADDRESS

```
#define USBDEV_ARB_RWx_DR_ADDRESS (m) (USBDEV_ARB_RW1_DR_ADDRESS + ((m) * (0x40)))
```

Endpoint Data Register Data Register for Endpoint

### 6.37.2.9 USBDEV\_SIE\_EPx\_CNT0\_ADDRESS

```
#define USBDEV_SIE_EPx_CNT0_ADDRESS (USBDEV_SIE_EP1_CNT0_ADDRESS)
```

Non-control endpoint count register The Endpoint Count Register 0 (CNT0) is used for configuring endpoints one through eight.

### 6.37.2.10 USBDEV\_SIE\_EPx\_CNT0\_DATA\_COUNT\_MSB\_MASK

```
#define USBDEV_SIE_EPx_CNT0_DATA_COUNT_MSB_MASK (0x00000007) /* <0:2> RW:RW:0: */
```

These bits are the 3 MSb bits of an 11-bit counter. The LSb are the Data Count[7:0] bits of the CNT1 register. Refer to the CNT1 register for more information.

### 6.37.2.11 USBDEV\_SIE\_EPx\_CNT0\_DATA\_TOGGLE

```
#define USBDEV_SIE_EPx_CNT0_DATA_TOGGLE (1u << 7) /* <7:7> RW:RW:0: */
```

This bit selects the DATA packet's toggle state. For IN transactions firmware must set this bit to the expected state. For OUT transactions the hardware sets this bit to the state of the received Data Toggle bit.

### 6.37.2.12 USBDEV\_SIE\_EPx\_CNT0\_DATA\_VALID

```
#define USBDEV_SIE_EPx_CNT0_DATA_VALID (1u << 6) /* <6:6> RW1S:RW0C:0: */
```

This bit is used for OUT transactions only and is read only. It is cleared to '0' if CRC bit stuffing errors or PID errors occur. This bit does not update for some endpoint mode settings.

### 6.37.2.13 USBDEV\_SIE\_EPx\_CNT1\_ADDRESS

```
#define USBDEV_SIE_EPx_CNT1_ADDRESS (USBDEV_SIE_EP1_CNT1_ADDRESS)
```

The Endpoint Count Register 1 (CNT1) sets or reports the number of bytes in a USB data transfer to the non-control endpoints. For IN transactions firmware loads the count with the number of data bytes to transmit to the host. Valid values for MODE 1 and MODE 2 are 0 to 514 and for MODE 3 it is 0 to 1025. For an OUT transaction the full 11-bit count is updated by the SIE to the actual number of data bytes received by the SIE plus two for the packet's CRC.

The bytes (Data + CRC) received both the data from the USB packet and the 2-byte CRC are written to the USB's dedicated SRAM. Valid values for MODE 1 and MODE 2 are 2 to 514 and for MODE 3 it is 2 to 1025. To get the actual number of bytes received firmware needs to decrement the 11-bit count by two.

#### 6.37.2.14 USBDEV\_SIE\_EPx\_CNT1\_DATA\_COUNT\_MASK

```
#define USBDEV_SIE_EPx_CNT1_DATA_COUNT_MASK (0x000000ff) /* <0:7> RW:RW:0: */
```

These bits are the 8 LSb of a 11-bit counter. The 3 MSb bits are in the CNT0 register. The 11-bit count indicates the number of data bytes in a transaction.

#### 6.37.2.15 USBDEV\_SIE\_EPx\_CR0\_ACKED\_TXN

```
#define USBDEV_SIE_EPx_CR0_ACKED_TXN (1u << 4) /* <4:4> RW1S:RWC:0: */
```

The ACK'd transaction bit is set whenever the SIE engages in a transaction to the register's endpoint that completes with an ACK packet. This bit is cleared by any writes to the register.

#### 6.37.2.16 USBDEV\_SIE\_EPx\_CR0\_ADDRESS

```
#define USBDEV_SIE_EPx_CR0_ADDRESS (USBDEV_SIE_EPx_CR0_ADDRESS)
```

Non-control endpoint's control Register The Endpoint Control Register 0 (CR0) is used for status and configuration of the non-control endpoints .

#### 6.37.2.17 USBDEV\_SIE\_EPx\_CR0\_ERR\_IN\_TXN

```
#define USBDEV_SIE_EPx_CR0_ERR_IN_TXN (1u << 6) /* <6:6> RW1S:RWC:0: */
```

The Error in transaction bit is set whenever an error is detected. For an IN transaction, this indicates a no response from HOST scenario. For an OUT transaction, this represents an RxErr (PID error/ CRC error/ bit-stuff error scenario). This bit is cleared by any writes to the register.

#### 6.37.2.18 USBDEV\_SIE\_EPx\_CR0\_MODE\_MASK

```
#define USBDEV_SIE_EPx_CR0_MODE_MASK (0x0000000f) /* <0:3> RW:RW:0: */
```

The mode controls how the USB SIE responds to traffic and how the USB SIE changes the mode of that endpoint as a result of host packets to the endpoint.

#### 6.37.2.19 USBDEV\_SIE\_EPx\_CR0\_NAK\_INT\_EN

```
#define USBDEV_SIE_EPx_CR0_NAK_INT_EN (1u << 5) /* <5:5> R:RW:0: */
```

When set this bit causes an endpoint interrupt to be generated even when a transfer completes with a NAK.

#### 6.37.2.20 USBDEV\_SIE\_EPx\_CR0\_STALL

```
#define USBDEV_SIE_EPx_CR0_STALL (1u << 7) /* <7:7> R:RW:0: */
```

When this bit is set the SIE stalls an OUT packet if the Mode bits are set to ACK-OUT. The SIE stalls an IN packet if the mode bits are set to ACK-IN. This bit must be clear for all other modes.

### 6.37.3 Typedef Documentation

### 6.37.3.1 usb\_event\_cb\_t

```
typedef void(* usb_event_cb_t) (usb_state_t state, uint32_t data)
```

Event callback function pointer.

The callback shall be invoked by the USB module if any USB state change occurs.

### 6.37.3.2 usb\_setup\_cb\_t

```
typedef ccg_status_t(* usb_setup_cb_t) (usb_setup_pkt_t *setup_pkt)
```

Setup callback function pointer.

Various types of setup function callback pointers can be registered. The following data type defines the function prototype for the setup callback function. The callback function is expected to return error codes as to how the request has handled. If the status is SUCCESS, the USB module stops handling the request internally. If the status is NOT\_SUPPORTED, then the module shall try to handle the request internally. If any other error is returned, it shall stall EP0.

Return value expectation: CCG\_STAT\_SUCCESS = The setup packet is successfully handled. CCG\_STAT\_NO\_T\_SUPPORTED = The setup packet was not handled. Any other value is treated as error and handled accordingly.

## 6.37.4 Enumeration Type Documentation

### 6.37.4.1 usb\_ep0\_state\_t

```
enum usb_ep0_state_t
```

USB EP0 states.

Enumerator

|                          |                              |
|--------------------------|------------------------------|
| USB_EP0_STATE_DISABLED   | EP0 is disabled.             |
| USB_EP0_STATE_SETUP      | EP0 in setup phase.          |
| USB_EP0_STATE_DATA_IN    | EP0 in data IN phase.        |
| USB_EP0_STATE_DATA_OUT   | EP0 in data OUT phase.       |
| USB_EP0_STATE_STATUS_IN  | EP0 in status IN ZLP phase.  |
| USB_EP0_STATE_STATUS_OUT | EP0 in status OUT ZLP phase. |
| USB_EP0_STATE_STALL      | EP0 in stall phase.          |

### 6.37.4.2 usb\_ep\_index\_t

```
enum usb_ep_index_t
```

Endpoint identifier.

Endpoint number is converted to zero indexed. EP1 is treated as 0. This avoids the subtraction requirement to create the indexed value.

Enumerator

|                  |                |
|------------------|----------------|
| USB_EP_INDEX_EP1 | EP1 identifier |
| USB_EP_INDEX_EP2 | EP2 identifier |

**Enumerator**

|                  |                |
|------------------|----------------|
| USB_EP_INDEX_EP3 | EP3 identifier |
| USB_EP_INDEX_EP4 | EP4 identifier |
| USB_EP_INDEX_EP5 | EP5 identifier |
| USB_EP_INDEX_EP6 | EP6 identifier |
| USB_EP_INDEX_EP7 | EP7 identifier |
| USB_EP_INDEX_EP8 | EP8 identifier |

**6.37.4.3 usb\_state\_t**

```
enum usb_state_t
```

USB device mode states.

**Enumerator**

|                         |                                           |
|-------------------------|-------------------------------------------|
| USB_STATE_DEINITED      | Un-initialized state.                     |
| USB_STATE_DISABLED      | USB module is disabled.                   |
| USB_STATE_WAIT_FOR_VBUS | Waiting for connection.                   |
| USB_STATE_CONNECTED     | Connected to an external USB host.        |
| USB_STATE_RESET         | When the device has received a reset.     |
| USB_STATE_ADDRESSED     | Device address has been set.              |
| USB_STATE_CONFIGURED    | A valid device configuration is selected. |
| USB_STATE_SUSPENDED     | USB is in suspended condition.            |
| USB_STATE_RESUMED       | Virtual state for the sake of callback.   |
| USB_STATE_UNCONFIGURED  | Virtual state for the sake of callback.   |

**6.37.4.4 usbdev\_ep\_mode\_t**

```
enum usbdev_ep_mode_t
```

USB endpoint modes The enumeration lists all the possible register configuration for endpoints states. This list is dependant on the hardware and should not be changed.

**6.37.5 Function Documentation****6.37.5.1 usb\_disable()**

```
ccg_status_t usb_disable (
 void
)
```

Disables the USB device mode of operation.

The function disables the USB hardware and disconnects the D+/D- lines.

**Returns**

Status of the call

**6.37.5.2 usb\_enable()**

```
ccg_status_t usb_enable (
 bool reg_enable)
```

Enables the USB device mode of operation.

The function initializes the USB hardware and enables the D+/D- lines and does a pull up on the D+ line for the external host to detect the presence of the device. The API expects that the USB block is already initialized.

**Parameters**

|                |                                                 |
|----------------|-------------------------------------------------|
| <i>Whether</i> | to enable the internal regulator for the block. |
|----------------|-------------------------------------------------|

**Returns**

Status of the call

**6.37.5.3 usb\_ep0\_send\_recv\_status()**

```
ccg_status_t usb_ep0_send_recv_status (
 void)
```

Sends / receives EP0 ZLP status phase.

The API completes the status phase of the current EP0 request. The caller is expected to call the function in sequence. The status phase is handled implicitly when the setup\_read and setup\_write functions are invoked correctly with the last flag set to true. The function does not wait for the transfer to complete. In case of an error, the EP0 shall be stalled by the USB module.

**Returns**

Status of the call

**6.37.5.4 usb\_ep0\_set\_stall()**

```
ccg_status_t usb_ep0_set_stall (
 void)
```

Stall EP0.

The function stalls endpoint zero to indicate error to current request. The stall is automatically cleared on receiving a new setup request.

**Returns**

Status of the call

### 6.37.5.5 `usb_ep0_setup_read()`

```
ccg_status_t usb_ep0_setup_read (
 uint8_t * data,
 uint16_t length,
 bool last,
 usb_setup_cb_t cb)
```

Setup a read data transfer on EP0.

The API does not wait for the read to complete. The function just updates the state machine and queues the first packet read. The read has to be completed by repeatedly queueing packet read requests. The last parameter can be used to do multiple partial transfers. For default single transfers, the last parameter should always be true.

#### Parameters

|               |                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>data</i>   | Buffer to write the received USB EP0 data. The caller should ensure that the buffer is capable of receiving upto a size of (length)                    |
| <i>length</i> | Length of data to be transfer. This has to be a multiple of eight bytes.                                                                               |
| <i>last</i>   | Whether the request is a partial transfer or not. Set to true if the module needs to implicitly handle the status phase after completing the transfer. |
| <i>cb</i>     | Callback to be invoked at the end of transfer. Optional.                                                                                               |

#### Returns

Status of the call

### 6.37.5.6 `usb_ep0_setup_write()`

```
ccg_status_t usb_ep0_setup_write (
 uint8_t * data,
 uint16_t length,
 bool last,
 usb_setup_cb_t cb)
```

Setup a write data transfer on EP0. The API does not wait for the write to complete. The function just updates the state machine and queues the first packet. The write has to be completed by repeatedly queueing packet requests. The last parameter can be used to do multiple partial transfers. For default single transfers, the last parameter should always be true.

#### Parameters

|               |                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>data</i>   | Buffer to write the USB EP0 data.                                                                                                                      |
| <i>length</i> | Length of data to be transfer.                                                                                                                         |
| <i>last</i>   | Whether the request is a partial transfer or not. Set to true if the module needs to implicitly handle the status phase after completing the transfer. |
| <i>cb</i>     | Callback to be invoked at the end of transfer. Optional.                                                                                               |

#### Returns

Status of the call

### 6.37.5.7 `usb_ep0_wait_for_ack()`

```
ccg_status_t usb_ep0_wait_for_ack (
 void)
```

Waits for the EP0 status phase to complete.

The function does a blocking wait until the status phase is completed. This function should be invoked only if the task loop can be safely blocked. Otherwise use the callback functionality available with the transfers.

#### Returns

Status of the call

### 6.37.5.8 `usb_ep_clear_stall()`

```
ccg_status_t usb_ep_clear_stall (
 usb_ep_index_t ep_index)
```

Clear the stall on the selected endpoint.

The function shall clear a previously stalled endpoint. The function shall reset the data toggle even if the endpoint was not previously stalled. This call shall also reset the endpoint to the default state. So read should be explicitly invoked for an OUT endpoint. The endpoint shall start NAKing all the IN / OUT tokens once the function gets executed.

#### Parameters

|                       |                             |
|-----------------------|-----------------------------|
| <code>ep_index</code> | Endpoint to clear the stall |
|-----------------------|-----------------------------|

#### Returns

Status of the call

### 6.37.5.9 `usb_ep_disable()`

```
ccg_status_t usb_ep_disable (
 usb_ep_index_t ep_index)
```

Disables a previously enabled endpoint.

The endpoint shall be disabled and all data in the FIFO cleared. The endpoint shall stop responding to requests from USB host.

#### Parameters

|                       |                         |
|-----------------------|-------------------------|
| <code>ep_index</code> | Endpoint to be disabled |
|-----------------------|-------------------------|

#### Returns

Status of the call

### 6.37.5.10 usb\_ep\_enable()

```
ccg_status_t usb_ep_enable (
 usb_ep_index_t ep_index,
 bool is_out)
```

Enable an endpoint with the selected configuration.

The endpoint shall be initialized and configured with the provided parameters. The API expects the endpoint to be in disabled state.

#### Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>ep_index</i> | Endpoint to be enabled - Caller needs to ensure validity |
| <i>is_out</i>   | Direction of the endpoint                                |

#### Returns

Status of the call

### 6.37.5.11 usb\_ep\_flush()

```
ccg_status_t usb_ep_flush (
 usb_ep_index_t ep_index)
```

Flushes and clears the selected endpoint.

The function resets the endpoint and re-arms an OUT endpoint to receive data if a receive was already queued.

#### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>ep_index</i> | Endpoint to be selected |
|-----------------|-------------------------|

#### Returns

Status of the call

### 6.37.5.12 usb\_ep\_is\_ready()

```
bool usb_ep_is_ready (
 usb_ep_index_t ep_index)
```

Checks whether the endpoint is ready for data transfer.

The API expects that the endpoint is enabled and active

#### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>ep_index</i> | Endpoint to be checked |
|-----------------|------------------------|

**Returns**

Endpoint status: true - If this is an IN endpoint, then the EP is ready to send data. If there was a previous transfer then it has completed successfully. If this is an OUT endpoint, then the EP has received a packet of data from the USB host. false - If this is an IN endpoint, data is being sent out and not completed. If this is an OUT endpoint, then the data is not yet received. If the endpoint is not active or USB connection is not active, this API returns CyFalse.

**6.37.5.13 usb\_ep\_queue\_read\_single()**

```
ccg_status_t usb_ep_queue_read_single (
 usb_ep_index_t ep_index)
```

Queues a read operation on the selected endpoint.

The function enables the selected endpoint to receive one packet of data. It does not wait for the data to be received.

**Parameters**

|                 |                         |
|-----------------|-------------------------|
| <i>ep_index</i> | Endpoint to be selected |
|-----------------|-------------------------|

**Returns**

Status of the call

**6.37.5.14 usb\_ep\_read\_single()**

```
ccg_status_t usb_ep_read_single (
 usb_ep_index_t ep_index,
 uint8_t * data,
 uint8_t * count)
```

Retrieves the data packet available on the endpoint.

The function expects that a data is already available and retrieves the packet from the endpoint buffer.

**Parameters**

|                 |                                                     |
|-----------------|-----------------------------------------------------|
| <i>ep_index</i> | Endpoint to be selected                             |
| <i>data</i>     | Buffer pointer to read the data into                |
| <i>count</i>    | Pointer to return the actual count of data received |

**Returns**

Status of the call

**6.37.5.15 usb\_ep\_send\_zlp()**

```
ccg_status_t usb_ep_send_zlp (
 usb_ep_index_t ep_index)
```

Send ZLP on the selected endpoint.

The function sends a zero-length packet on the selected IN endpoint.

#### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>ep_index</i> | Endpoint to be selected |
|-----------------|-------------------------|

#### Returns

Status of the call

### 6.37.5.16 `usb_ep_set_stall()`

```
ccg_status_t usb_ep_set_stall (
 usb_ep_index_t ep_index)
```

Stall the selected endpoint.

The endpoint shall stall all IN / OUT tokens after the function has been executed.

#### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>ep_index</i> | Endpoint to be stalled |
|-----------------|------------------------|

#### Returns

Status of the call

### 6.37.5.17 `usb_ep_write_single()`

```
ccg_status_t usb_ep_write_single (
 usb_ep_index_t ep_index,
 uint8_t * data,
 uint8_t count)
```

Queue a packet on the selected IN endpoint.

The function copies the data available in the buffer to the endpoint buffer and arms the endpoint for transfer.

#### Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>ep_index</i> | Endpoint to be selected                                  |
| <i>data</i>     | Buffer pointer where the data is available               |
| <i>count</i>    | Numbers of bytes to transmit. The size cannot exceed 64. |

#### Returns

Status of the call

#### 6.37.5.18 usb\_get\_state()

```
usb_state_t usb_get_state (
 void)
```

Get current USB state.

The API returns the current state of the USB device module.

##### Returns

Current USB state

#### 6.37.5.19 usb\_init()

```
ccg_status_t usb_init (
 usb_config_t * cfg)
```

Initialize the USB module.

The API initializes the USB interface. This is mainly a software state machine initialization. The PHY is not enabled at this point. The API helps to cleanup previous state information.

##### Parameters

|            |                                      |
|------------|--------------------------------------|
| <i>cfg</i> | USB module configuration parameters. |
|------------|--------------------------------------|

##### Returns

Status of the call

#### 6.37.5.20 usb\_intr\_lock()

```
uint32_t usb_intr_lock (
 void)
```

The function locks all USB interrupts to provide USB module specific mutex lock.

The function allows to lock all USB module interrupts without having to lock all interrupts in the system. It should be noted that USB module cannot be started or stopped from a locked state. The function can be called in a nested fashion. But it should be noted that the unlock should be done in the reverse order of locking to ensure correct state on exit.

##### Returns

32-bit lock status at the time of the lock.

#### 6.37.5.21 usb\_intr\_unlock()

```
void usb_intr_unlock (
 uint32_t lock)
```

The function un-locks USB interrupts due to a previous lock call.

The function can be called in a nested fashion. But it should be noted that the unlock should be done in the reverse order of locking to ensure correct state on exit.

#### Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>lock</i> | Lock state returned the previous lock call |
|-------------|--------------------------------------------|

#### Returns

None

### 6.37.5.22 `usb_is_idle()`

```
bool usb_is_idle (
 void)
```

Check whether the USB module is idle or not.

The API checks if there are any pending task for USB module. The function is expected to be invoked before entering SLEEP mode. It should be noted that the DEEP\_SLEEP mode requires USB block to be in suspend state and this has been checked using [usb\\_get\\_state\(\)](#) function.

#### Returns

true = USB module is idle, false = USB module is busy.

### 6.37.5.23 `usb_remote_wakeup()`

```
ccg_status_t usb_remote_wakeup (
 void)
```

Sends a remote wakeup signal on the USB bus.

The API sends a remote wakeup event on the USB bus if the USB bus is in suspended state and if the host has requested for a remote wakeup.

#### Returns

Status of the call.

### 6.37.5.24 `usb_task()`

```
void usb_task (
 void)
```

USB module task handler.

USB module interrupts have considerable processing and the task handler is used to deferring the handling to main task loop. The function is expected to be invoked continuously from the main task loop. It is expected that the loop gets to run at least once every millisecond whenever there is activity.

#### Returns

None

### 6.37.5.25 `usb_vbus_int_handler()`

```
void usb_vbus_int_handler (
 bool vbus_state)
```

VBUS state change handler function.

The function is expected to be invoked from by the application code whenever VBUS state changes. For a USB device, the VBUS turning on is a connect event and VBUS turning off is a disconnect event. The USB block control shall be done accordingly.

#### Parameters

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <code>vbus_state</code> | Current VBUS state. true = VBUS present, false = VBUS absent. |
|-------------------------|---------------------------------------------------------------|

#### Returns

None

## 6.37.6 Variable Documentation

### 6.37.6.1 USBARB

```
const PUSBARB_REGS_T USBARB[]
```

Register structure extern

### 6.37.6.2 USBARB16

```
const PUSBARB16_REGS_T USBARB16[]
```

Register structure extern

### 6.37.6.3 USBSIE

```
const PUSBSIE_REGS_T USBSIE[]
```

Register structure extern

## 6.38 usb/usbconst.h File Reference

### Data Structures

- struct [usb\\_setup\\_pkt\\_t](#)

### Macros

- #define **USB\_TARGET\_MASK** (0x03)
- #define **USB\_TARGET\_DEVICE** (0x00)
- #define **USB\_TARGET\_INTF** (0x01)
- #define **USB\_TARGET\_ENDPT** (0x02)
- #define **USB\_TARGET\_OTHER** (0x03)

- #define **USB\_GS\_DEVICE** (0x80)
- #define **USB\_GS\_INTERFACE** (0x81)
- #define **USB\_GS\_ENDPOINT** (0x82)
- #define **USB\_SETUP\_PKT\_DIRECTION** (0x80)
- #define **USB\_TYPE\_MASK** (0x60)
- #define **USB\_STANDARD\_RQT** (0x00)
- #define **USB\_CLASS\_RQT** (0x20)
- #define **USB\_VENDOR\_RQT** (0x40)
- #define **USB\_RESERVED\_RQT** (0x60)
- #define **USB\_CLASS\_HID** (0x03)
- #define **USB\_CLASS\_BILLBOARD** (0x11)
- #define **USB\_CLASS\_TYPEC\_BRIDGE** (0x12)
- #define **USB\_CLASS\_VENDOR** (0xFF)
- #define **USB\_MAX\_DEVICE\_ADDR** (127)
- #define **USB\_CRC\_SIZE** (2)
- #define **USB\_SETUP\_PKT\_SIZE** (8)
- #define **USB\_DEVICE\_DSCR\_SIZE** (18)
- #define **USB\_CONFIG\_DSCR\_SIZE** (9)
- #define **USB\_INF\_DSCR\_SIZE** (9)
- #define **USB\_HID\_DSCR\_SIZE** (9)
- #define **USB\_EP\_DSCR\_SIZE** (7)
- #define **USB\_BOS\_DSCR\_SIZE** (5)
- #define **USB\_BOS\_USB2\_EXTN\_DSCR\_SIZE** (7)
- #define **USB\_BOS\_CONTAINER\_ID\_DSCR\_SIZE** (20)
- #define **USB\_ALT\_MODE\_CAP\_DSCR\_SIZE** (8)
- #define **USB\_DEVICE\_DSCR\_VID\_OFFSET** (8)
- #define **USB\_DEVICE\_DSCR\_PID\_OFFSET** (10)
- #define **USB\_DEVICE\_DSCR\_BCD\_DEV\_OFFSET** (12)
- #define **USB\_DEVICE\_DSCR\_SERIAL\_OFFSET** (16)
- #define **USB\_CONFIG\_DSCR\_NUM\_INF\_OFFSET** (4)
- #define **USB\_CONFIG\_DSCR\_ATTRIB\_OFFSET** (7)
- #define **USB\_CONFIG\_DSCR\_MAXPOWER\_OFFSET** (8)
- #define **USB\_CONFIG\_DSCR\_SELF\_POWERED** (0x40)

## Enumerations

- enum **usb\_setup\_cmd\_t** {
 **USB\_SC\_GET\_STATUS** = 0x00, **USB\_SC\_CLEAR\_FEATURE**, **USB\_SC\_RESERVED**, **USB\_SC\_SET\_FEATURE**,
 **USB\_SC\_SET\_ADDRESS** = 0x05, **USB\_SC\_GET\_DESCRIPTOR**, **USB\_SC\_SET\_DESCRIPTOR**,
 **USB\_SC\_GET\_CONFIGURATION**,
 **USB\_SC\_SET\_CONFIGURATION**, **USB\_SC\_GET\_INTERFACE**, **USB\_SC\_SET\_INTERFACE**, **USB\_SC\_SYNC\_FRAME**,
 **USB\_SC\_SET\_SEL** = 0x30, **USB\_SC\_SET\_ISOC\_DELAY** }
- enum **usb\_ep\_type\_t** { **USB\_EP\_CONTROL** = 0, **USB\_EP\_ISO** = 1, **USB\_EP\_BULK** = 2, **USB\_EP\_INTR** = 3 }
- enum **usb\_dscr\_type\_t** {
 **USB\_DEVICE\_DSCR** = 0x01, **USB\_CONFIG\_DSCR**, **USB\_STRING\_DSCR**, **USB\_INF\_DSCR**,
 **USB\_EP\_DSCR**, **USB\_DEVQUAL\_DSCR**, **USB\_OTHERSPEED\_DSCR**, **USB\_INF\_POWER\_DSCR**,
 **USB\_BOS\_DSCR** = 0x0F, **USB\_DEVICE\_CAPB\_DSCR**, **USB\_HID\_DSCR** = 0x21, **USB\_REPORT\_DSCR** }
- enum **usb\_dev\_cap\_type\_t** {
 **USB\_CAP\_TYPE\_WIRELESS** = 0x01, **USB\_CAP\_TYPE\_USB2\_EXTN**, **USB\_CAP\_TYPE\_SS\_USB**,
 **USB\_CAP\_TYPE\_CONTAINER\_ID**,
 **USB\_CAP\_TYPE\_BILLBOARD** = 0x0D }
- enum **usb\_feature\_select\_t** { **USB\_EP\_HALT** = 0, **USB\_REMOTE\_WAKE** = 1, **USB\_TEST\_MODE** = 2 }
- enum **usb\_hid\_rqt\_t** {
 **USB\_HID\_RQT\_GET\_REPORT** = 0x01, **USB\_HID\_RQT\_GET\_IDLE** = 0x02, **USB\_HID\_RQT\_GET\_PROTOCOL** = 0x03,
 **USB\_HID\_RQT\_SET\_REPORT** = 0x09,
 **USB\_HID\_RQT\_SET\_IDLE** = 0x0A, **USB\_HID\_RQT\_SET\_PROTOCOL** = 0x0B }

### 6.38.1 Detailed Description

Constants defined in the USB specification.

### 6.38.2 Macro Definition Documentation

#### 6.38.2.1 USB\_TARGET\_MASK

```
#define USB_TARGET_MASK (0x03)
```

USB Setup Commands Recipient

### 6.38.3 Enumeration Type Documentation

#### 6.38.3.1 usb\_dev\_cap\_type\_t

```
enum usb_dev_cap_type_t
```

Enumeration of device capability types.

The BOS descriptor supports various device capability descriptors. The enumeration contains the various types for capability descriptors.

Enumerator

|                           |                                       |
|---------------------------|---------------------------------------|
| USB_CAP_TYPE_WIRELESS     | Wireless capability descriptor        |
| USB_CAP_TYPE_USB2_EXTN    | USB 2.0 extension descriptor          |
| USB_CAP_TYPE_SS_USB       | USB super-speed capability descriptor |
| USB_CAP_TYPE_CONTAINER_ID | Container ID capability descriptor    |
| USB_CAP_TYPE_BILLBOARD    | Billboard capability descriptor       |

#### 6.38.3.2 usb\_dscr\_type\_t

```
enum usb_dscr_type_t
```

Enumeration of descriptor types.

A USB device is identified by the descriptors provided. The following are the standard defined descriptors.

Enumerator

|                     |                                      |
|---------------------|--------------------------------------|
| USB_DEVICE_DSCR     | Super speed device descriptor        |
| USB_CONFIG_DSCR     | Configuration descriptor             |
| USB_STRING_DSCR     | String descriptor                    |
| USB_INF_DSCR        | Interface descriptor                 |
| USB_EP_DSCR         | Endpoint descriptor                  |
| USB_DEVQUAL_DSCR    | Device qualifier descriptor          |
| USB_OTHERSPEED_DSCR | Other speed configuration descriptor |

**Enumerator**

|                                   |                              |
|-----------------------------------|------------------------------|
| <code>USB_INF_POWER_DSCR</code>   | Interface power descriptor   |
| <code>USB_BOS_DSCR</code>         | BOS descriptor               |
| <code>USB_DEVICE_CAPB_DSCR</code> | Device capability descriptor |
| <code>USB_HID_DSCR</code>         | HID descriptor               |
| <code>USB_REPORT_DSCR</code>      | Report descriptor            |

**6.38.3.3 `usb_ep_type_t`**

```
enum usb_ep_type_t
```

Enumeration of the endpoint types.

There are four types of endpoints. This defines the behaviour of the endpoint. The control endpoint is a compulsory for any device whereas the other endpoints are used as per requirement.

**Enumerator**

|                             |                           |
|-----------------------------|---------------------------|
| <code>USB_EP_CONTROL</code> | Control Endpoint Type     |
| <code>USB_EP_ISO</code>     | Isochronous Endpoint Type |
| <code>USB_EP_BULK</code>    | Bulk Endpoint Type        |
| <code>USB_EP_INTR</code>    | Interrupt Endpoint Type   |

**6.38.3.4 `usb_feature_select_t`**

```
enum usb_feature_select_t
```

List of USB feature selector codes.

The following are the various features that can be selected using the SetFeature request or cleared using Clear← Feature setup request. Refer to the USB specification for more information.

**Enumerator**

|                              |                                                     |
|------------------------------|-----------------------------------------------------|
| <code>USB_EP_HALT</code>     | USB Endpoint HALT feature. Sets or clears EP stall. |
| <code>USB_REMOTE_WAKE</code> | USB 2.0 Remote Wakeup.                              |
| <code>USB_TEST_MODE</code>   | USB 2.0 Test mode.                                  |

**6.38.3.5 `usb_setup_cmd_t`**

```
enum usb_setup_cmd_t
```

Standard device request codes.

These are the various standard requests received from the USB host. The device is expected to respond to all these requests.

## Enumerator

|                          |                                    |
|--------------------------|------------------------------------|
| USB_SC_GET_STATUS        | Get status request.                |
| USB_SC_CLEAR_FEATURE     | Clear feature.                     |
| USB_SC_RESERVED          | Reserved command.                  |
| USB_SC_SET_FEATURE       | Set feature.                       |
| USB_SC_SET_ADDRESS       | Set address.                       |
| USB_SC_GET_DESCRIPTOR    | Get descriptor.                    |
| USB_SC_SET_DESCRIPTOR    | Set descriptor.                    |
| USB_SC_GET_CONFIGURATION | Get configuration.                 |
| USB_SC_SET_CONFIGURATION | Set configuration.                 |
| USB_SC_GET_INTERFACE     | Get interface (alternate setting). |
| USB_SC_SET_INTERFACE     | Set interface (alternate setting). |
| USB_SC_SYNC_FRAME        | Synch frame.                       |
| USB_SC_SET_SEL           | Set system exit latency.           |
| USB_SC_SET_ISOC_DELAY    | Set isochronous delay.             |



# Index

act\_cbl\_vdo  
    pd\_do\_t, 85  
act\_link\_train  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 144  
active\_alt\_inf  
    usb\_handle\_t, 136  
active\_boot\_app  
    sys\_fw\_metadata\_t, 132  
active\_cbl  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 144  
active\_cfg  
    usb\_handle\_t, 136  
afc\_src\_cap\_cnt  
    chg\_cfg\_params\_t, 47  
afc\_src\_caps  
    chg\_cfg\_params\_t, 47  
alert  
    dpm\_status\_t, 56  
alt\_mode  
    alt\_mode\_evt\_t::ALT\_MODE\_EVT, 21  
alt\_mode\_app\_cbk\_t  
    alt\_modes\_mngr.h, 162  
alt\_mode\_app\_cmd\_t  
    alt\_modes\_mngr.h, 163  
alt\_mode\_app\_evt\_t  
    alt\_modes\_mngr.h, 163  
alt\_mode\_cbk\_t  
    alt\_modes\_mngr.h, 163  
alt\_mode\_entered  
    app\_status\_t, 34  
alt\_mode\_event  
    alt\_mode\_evt\_t, 23  
alt\_mode\_event\_data  
    alt\_mode\_evt\_t, 23  
alt\_mode\_evt  
    alt\_mode\_evt\_t::ALT\_MODE\_EVT, 21  
alt\_mode\_evt\_t, 22  
    alt\_mode\_event, 23  
    alt\_mode\_event\_data, 23  
    val, 23  
alt\_mode\_evt\_t::ALT\_MODE\_EVT\_DATA, 22  
    evt\_data, 22  
    evt\_type, 22  
alt\_mode\_evt\_t::ALT\_MODE\_EVT, 21  
    alt\_mode, 21  
    alt\_mode\_evt, 21  
    data\_role, 21  
    svid, 21  
alt\_mode\_get\_status  
    alt\_modes\_mngr.h, 165  
alt\_mode\_hw.h  
    alt\_mode\_hw\_deinit, 157  
    alt\_mode\_hw\_is\_idle, 157  
    alt\_mode\_hw\_set\_cbk, 157  
    alt\_mode\_hw\_sleep, 158  
    alt\_mode\_hw\_t, 156  
    alt\_mode\_hw\_wakeup, 158  
    dp\_snk\_get\_hpd\_state, 158  
    eval\_app\_alt\_hw\_cmd, 159  
    eval\_hpd\_cmd, 159  
    eval\_mux\_cmd, 159  
    get\_mux\_state, 160  
    mux\_poll\_status\_t, 156  
    mux\_select\_t, 156  
    set\_mux, 160  
alt\_mode\_hw\_deinit  
    alt\_mode\_hw.h, 157  
alt\_mode\_hw\_evt\_t, 24  
    hw\_evt, 24  
    val, 24  
alt\_mode\_hw\_evt\_t::ALT\_MODE\_HW\_EVT, 23  
    data\_role, 24  
    evt\_data, 24  
    hw\_type, 24  
alt\_mode\_hw\_is\_idle  
    alt\_mode\_hw.h, 157  
alt\_mode\_hw\_set\_cbk  
    alt\_mode\_hw.h, 157  
alt\_mode\_hw\_sleep  
    alt\_mode\_hw.h, 158  
alt\_mode\_hw\_t  
    alt\_mode\_hw.h, 156  
alt\_mode\_hw\_wakeup  
    alt\_mode\_hw.h, 158  
alt\_mode\_id  
    alt\_mode\_info\_t, 25  
    alt\_mode\_reg\_info\_t, 28  
    comp\_tbl\_t, 49  
alt\_mode\_info\_t, 25  
    alt\_mode\_id, 25  
    app\_evt\_data, 25  
    app\_evt\_needed, 25  
    cbk, 26  
    cbl\_obj\_pos, 26  
    custom\_att\_obj\_pos, 26  
    eval\_app\_cmd, 26  
    is\_active, 26  
    mode\_state, 26

obj\_pos, 26  
 set\_mux\_isolate, 26  
 sop\_state, 26  
 uvd़\_supp, 27  
 vdm\_header, 27  
 vdo, 27  
 vdo\_max\_numb, 27  
 vdo\_numb, 27  
 alt\_mode\_mngr\_exit\_all  
     alt\_modes\_mngr.h, 165  
 alt\_mode\_mngr\_sleep  
     alt\_modes\_mngr.h, 165  
 alt\_mode\_mngr\_state\_t  
     alt\_modes\_mngr.h, 164  
 alt\_mode\_mngr\_wakeup  
     alt\_modes\_mngr.h, 166  
 alt\_mode\_reg\_info\_t, 27  
     alt\_mode\_id, 28  
     app\_evt, 28  
     atch\_tgt\_info, 28  
     atch\_type, 28  
     cbl\_sop\_flag, 28  
     data\_role, 28  
     svid\_emca\_vdo, 28  
     svid\_vdo, 28  
 alt\_mode\_state\_t  
     alt\_modes\_mngr.h, 164  
 alt\_mode\_trig\_mask  
     app\_status\_t, 34  
 alt\_modes\_mngr.h  
     alt\_mode\_app\_cbk\_t, 162  
     alt\_mode\_app\_cmd\_t, 163  
     alt\_mode\_app\_evt\_t, 163  
     alt\_mode\_cbk\_t, 163  
     alt\_mode\_get\_status, 165  
     alt\_mode\_mngr\_exit\_all, 165  
     alt\_mode\_mngr\_sleep, 165  
     alt\_mode\_mngr\_state\_t, 164  
     alt\_mode\_mngr\_wakeup, 166  
     alt\_mode\_state\_t, 164  
     eval\_app\_alt\_mode\_cmd, 166  
     eval\_rec\_vdm, 166  
     fail\_status\_t, 164  
     form\_alt\_mode\_event, 167  
     get\_alt\_mode\_numb, 167  
     get\_vdm\_buff, 168  
     is\_alt\_mode\_mngr\_idle, 168  
     is\_svid\_supported, 168  
     reg\_alt\_mode\_mngr, 169  
     reset\_alt\_mode\_info, 169  
     vdm\_task\_mngr\_alt\_mode\_process, 169  
 alt\_status  
     bb\_handle\_t, 41  
 ama\_fw\_ver  
     pd\_do\_t::STD\_AMA\_VDO\_PD3, 123  
     pd\_do\_t::STD\_AMA\_VDO, 122  
 ama\_hw\_ver  
     pd\_do\_t::STD\_AMA\_VDO\_PD3, 123  
     pd\_do\_t::STD\_AMA\_VDO, 122  
     pd.h, 284  
 app.h  
     app\_conf\_for\_faulty\_dev\_removal, 185  
     app\_disable\_pd\_port, 185  
     app\_event\_handler, 186  
     app\_get\_callback\_ptr, 186  
     app\_get\_resp\_buf, 186  
     app\_get\_status, 187  
     app\_init, 187  
     app\_ovp\_disable, 187  
     app\_ovp\_enable, 187  
     app\_sleep, 188  
     app\_task, 188  
     app\_update\_bc\_src\_support, 188  
     app\_update\_sys\_pwr\_state, 189  
     app\_uvp\_disable, 189  
     app\_uvp\_enable, 189  
     app\_validate\_configtable\_offsets, 191  
     app\_wakeup, 191  
     ccg\_app\_task\_init, 191  
     mux\_ctrl\_init, 192  
     mux\_ctrl\_set\_cfg, 192  
     mux\_poll\_fnc\_cbk\_t, 184  
     sln\_pd\_event\_handler, 192  
     sys\_hw\_error\_type\_t, 185  
     system\_sleep, 193  
     system\_vconn\_ocp\_dis, 193  
     system\_vconn\_ocp\_en, 193  
     vbus\_discharge\_off, 194  
     vbus\_discharge\_on, 194  
     vbus\_get\_value, 194  
     vbus\_is\_present, 195  
     vconn\_disable, 195  
     vconn\_enable, 195  
     vconn\_is\_present, 196  
     app/alt\_mode/alt\_mode\_hw.h, 155  
     app/alt\_mode/alt\_modes\_mngr.h, 161  
     app/alt\_mode/dp\_sid.h, 170  
     app/alt\_mode/intel\_ridge.h, 173  
     app/alt\_mode/intel\_vid.h, 176  
     app/alt\_mode/vdm\_task\_mngr.h, 178  
     app/app.h, 181  
     app/billboard.h, 196  
     app/pdo.h, 204  
     app/psink.h, 205  
     app/psource.h, 207  
     app/ridge\_slave/ridge\_slave.h, 209  
     app/swap.h, 213  
     app/usb\_hid.h, 214  
     app/usb\_i2cm.h, 220  
     app/uvdm.h, 223  
     app/vdm.h, 226  
     app\_cbk  
         dpm\_status\_t, 56

app\_cbk\_t, 29  
    app\_event\_handler, 29  
    eval\_dr\_swap, 30  
    eval\_fr\_swap, 30  
    eval\_pr\_swap, 30  
    eval\_rdo, 30  
    eval\_src\_cap, 30  
    eval\_vconn\_swap, 30  
    eval\_vdm, 30  
    psnk\_disable, 30  
    psnk\_enable, 30  
    psnk\_set\_current, 31  
    psnk\_set\_voltage, 31  
    psrc\_disable, 31  
    psrc\_enable, 31  
    psrc\_set\_current, 31  
    psrc\_set\_voltage, 31  
    vbus\_discharge\_off, 31  
    vbus\_discharge\_on, 31  
    vbus\_get\_value, 31  
    vbus\_is\_present, 32  
    vconn\_disable, 32  
    vconn\_enable, 32  
    vconn\_is\_present, 32  
app\_conf\_for\_faulty\_dev\_removal  
    app.h, 185  
app\_disable\_pd\_port  
    app.h, 185  
app\_event\_handler  
    app.h, 186  
    app\_cbk\_t, 29  
app\_evt  
    alt\_mode\_reg\_info\_t, 28  
app\_evt\_data  
    alt\_mode\_info\_t, 25  
app\_evt\_needed  
    alt\_mode\_info\_t, 25  
app\_evt\_t  
    pd.h, 285  
app\_fault\_mask\_t  
    pd.h, 286  
app\_get\_callback\_ptr  
    app.h, 186  
app\_get\_resp\_buf  
    app.h, 186  
app\_get\_status  
    app.h, 187  
app\_init  
    app.h, 187  
app\_ovp\_disable  
    app.h, 187  
app\_ovp\_enable  
    app.h, 187  
app\_req\_status\_t  
    pd.h, 286  
app\_resp  
    app\_status\_t, 34  
app\_resp\_cbk\_t  
    pd.h, 283  
app\_resp\_t, 32  
    req\_status, 32  
    resp\_do, 33  
app\_sleep  
    app.h, 188  
app\_status\_t, 33  
    alt\_mode\_entered, 34  
    alt\_mode\_trig\_mask, 34  
    app\_resp, 34  
    bc\_12\_src\_disabled, 34  
    cbl\_disc\_id\_finished, 34  
    cur\_fb\_enabled, 34  
    fault\_status, 34  
    is\_mux\_busy, 34  
    is\_vbus\_on, 34  
    is\_vconn\_on, 35  
    is\_vdm\_pending, 35  
    ld\_sw\_ctrl, 35  
    mux\_poll\_cbk, 35  
    psnk\_cur, 35  
    psnk\_volt, 35  
    psrc\_rising, 35  
    psrc\_volt, 35  
    psrc\_volt\_old, 35  
    pwr\_ready\_cbk, 36  
    snk\_dis\_cbk, 36  
    vdm\_prcs\_failed, 36  
    vdm\_resp, 36  
    vdm\_resp\_cbk, 36  
    vdm\_retry\_pending, 36  
    vdm\_task\_en, 36  
    vdm\_version, 36  
app\_swap\_resp\_t  
    pd.h, 286  
app\_task  
    app.h, 188  
app\_update\_bc\_src\_support  
    app.h, 188  
app\_update\_sys\_pwr\_state  
    app.h, 189  
app\_uvp\_disable  
    app.h, 189  
app\_uvp\_enable  
    app.h, 189  
app\_validate\_configtable\_offsets  
    app.h, 191  
app\_wakeup  
    app.h, 191  
apple\_src\_id  
    chg\_cfg\_params\_t, 48  
application  
    pd\_config\_t, 81  
atch\_tgt\_info  
    alt\_mode\_reg\_info\_t, 28  
atch\_tgt\_info\_t, 37  
    ama\_vdo, 37  
    cbl\_svid, 37

cbl\_vdo, 37  
 tgt\_id\_header, 37  
 tgt\_svid, 37  
 atch\_type  
     alt\_mode\_reg\_info\_t, 28  
 attach  
     dpm\_status\_t, 56  
 attached\_dev  
     dpm\_status\_t, 56  
 attrib  
     usb\_setup\_pkt\_t, 140  
 aux\_resistor\_config\_t  
     pdss\_hal.h, 339  
  
 BB\_MAX\_EP0\_XFER\_SIZE  
     billboard.h, 197  
 BC\_SINK\_1\_2\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SINK\_APPLE\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_1\_2\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_AFC\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_APPLE\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_QC\_4\_0\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_QC\_MODE\_ENABLE\_MASK  
     pd.h, 268  
 BC\_SRC\_QC\_VER\_2\_CLASS\_A\_VAL  
     pd.h, 268  
 BC\_SRC\_QC\_VER\_2\_CLASS\_B\_VAL  
     pd.h, 268  
 BC\_SRC\_QC\_VER\_3\_CLASS\_A\_VAL  
     pd.h, 269  
 BC\_SRC\_QC\_VER\_3\_CLASS\_B\_VAL  
     pd.h, 269  
 BDO\_HDR\_IDX  
     pd.h, 269  
 bat\_chg\_params\_t, 38  
     reserved\_0, 38  
     reserved\_1, 38  
     table\_len, 38  
     vbatt\_cutoff\_volt, 38  
     vbatt\_dischg\_en\_volt, 38  
     vbatt\_max\_cur, 38  
     vbatt\_max\_volt, 39  
 bat\_chg\_tbl\_offset  
     pd\_port\_config\_t, 96  
 bat\_snk  
     pd\_do\_t, 85  
 bat\_src  
     pd\_do\_t, 85  
 battery\_input  
     pd\_power\_status\_t, 107  
 bb\_add\_info  
     bb\_handle\_t, 41  
 bb\_alt\_mode\_status\_t  
     billboard.h, 198  
 bb\_always\_on  
     bb\_settings\_t, 43  
 bb\_bos\_dscr\_offset  
     bb\_settings\_t, 43  
 bb\_bridge\_ctrl  
     billboard.h, 199  
 bb\_bus\_power  
     bb\_settings\_t, 43  
 bb\_cause\_t  
     billboard.h, 198  
 bb\_disable  
     billboard.h, 200  
 bb\_ec\_present  
     bb\_settings\_t, 43  
 bb\_enable  
     bb\_settings\_t, 43  
     billboard.h, 200  
 bb\_enter\_deep\_sleep  
     billboard.h, 201  
 bb\_flashing\_ctrl  
     billboard.h, 201  
 bb\_get\_version  
     billboard.h, 201  
 bb\_handle\_t, 40  
     alt\_status, 41  
     bb\_add\_info, 41  
     ep0\_buffer, 41  
     flashing\_mode, 41  
     num\_alt\_modes, 41  
     queue\_disable, 41  
     queue\_enable, 41  
     queue\_i2cm\_enable, 42  
     state, 42  
     timeout, 42  
     type, 42  
     usb\_configured, 42  
     usb\_i2cm\_mode, 42  
     usb\_port, 42  
 bb\_init  
     billboard.h, 202  
 bb\_is\_idle  
     billboard.h, 202  
 bb\_is\_present  
     billboard.h, 202  
 bb\_option  
     bb\_settings\_t, 44  
 bb\_pid  
     bb\_settings\_t, 44  
 bb\_settings\_t, 43  
     bb\_always\_on, 43  
     bb\_bos\_dscr\_offset, 43  
     bb\_bus\_power, 43  
     bb\_ec\_present, 43  
     bb\_enable, 43  
     bb\_option, 44  
     bb\_pid, 44  
     bb\_string\_dscr\_offset, 44

bb\_timeout, 44  
bb\_unique\_container\_id, 44  
bb\_vid, 44  
reserved\_1, 44  
table\_len, 45  
bb\_state\_t  
    billboard.h, 198  
bb\_string\_dscr\_offset  
    bb\_settings\_t, 44  
bb\_task  
    billboard.h, 203  
bb\_tbl\_offset  
    pd\_port\_config\_t, 96  
bb\_timeout  
    bb\_settings\_t, 44  
bb\_type\_t  
    billboard.h, 198  
bb\_unique\_container\_id  
    bb\_settings\_t, 44  
bb\_update\_all\_status  
    billboard.h, 203  
bb\_update\_alt\_status  
    billboard.h, 203  
bb\_usb\_string\_index\_t  
    billboard.h, 199  
bb\_vid  
    bb\_settings\_t, 44  
bc\_12\_src\_disabled  
    app\_status\_t, 34  
bcd\_dev  
    pd\_do\_t::STD\_PROD\_VDO, 128  
billboard.h  
    BB\_MAX\_EP0\_XFER\_SIZE, 197  
    bb\_alt\_mode\_status\_t, 198  
    bb\_bridge\_ctrl, 199  
    bb\_cause\_t, 198  
    bb\_disable, 200  
    bb\_enable, 200  
    bb\_enter\_deep\_sleep, 201  
    bb\_flashing\_ctrl, 201  
    bb\_get\_version, 201  
    bb\_init, 202  
    bb\_is\_idle, 202  
    bb\_is\_present, 202  
    bb\_state\_t, 198  
    bb\_task, 203  
    bb\_type\_t, 198  
    bb\_update\_all\_status, 203  
    bb\_update\_alt\_status, 203  
    bb\_usb\_string\_index\_t, 199  
bist\_cm2\_enabled  
    dpm\_status\_t, 56  
bist\_do  
    pd\_do\_t, 85  
bist\_mode\_t  
    pd.h, 287  
boot.h  
    boot\_check\_for\_valid\_fw, 377  
    boot\_get\_boot\_seq, 377  
    boot\_get\_wait\_time, 378  
    boot\_handle\_validate\_fw\_cmd, 378  
    boot\_jump\_to\_fw, 378  
    boot\_start, 378  
    boot\_update\_fw\_status, 379  
    boot\_validate\_configtable, 379  
    boot\_validate\_fw, 379  
    get\_boot\_mode\_reason, 380  
boot\_app\_id  
    sys\_fw\_metadata\_t, 132  
boot\_app\_ver\_status  
    sys\_fw\_metadata\_t, 132  
boot\_app\_version  
    sys\_fw\_metadata\_t, 132  
boot\_check\_for\_valid\_fw  
    boot.h, 377  
boot\_get\_boot\_seq  
    boot.h, 377  
boot\_get\_wait\_time  
    boot.h, 378  
boot\_handle\_validate\_fw\_cmd  
    boot.h, 378  
boot\_jump\_to\_fw  
    boot.h, 378  
boot\_last\_row  
    sys\_fw\_metadata\_t, 132  
boot\_mode\_request  
    fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
boot\_seq  
    sys\_fw\_metadata\_t, 132  
boot\_start  
    boot.h, 378  
boot\_update\_fw\_status  
    boot.h, 379  
boot\_validate\_configtable  
    boot.h, 379  
boot\_validate\_fw  
    boot.h, 379  
bootup  
    dpm\_status\_t, 57  
buf\_size  
    i2c\_scb\_config\_t, 74  
buffer  
    i2c\_scb\_config\_t, 74  
bus\_power  
    usb\_config\_t, 134  
CC\_CHANNEL\_1  
    pd.h, 269  
CC\_CHANNEL\_2  
    pd.h, 269  
CCG\_CC\_STAT\_DRP\_TOGGLE  
    pd.h, 269  
CCG\_CC\_STAT\_RD\_PRESENT  
    pd.h, 269  
CCG\_CC\_STAT\_RP\_PRESENT  
    pd.h, 269  
CCG\_CC\_STAT\_VCONN\_ACTIVE

pd.h, 269  
**CCG\_CC\_STAT\_ZOPEN**  
 pd.h, 270  
**CCG\_DPM\_ERROR\_NO\_VCONN**  
 pd.h, 270  
**CCG\_DPM\_ERROR\_NONE**  
 pd.h, 270  
**CCG\_FRS\_RX\_ENABLE\_MASK**  
 pd.h, 270  
**CCG\_FRS\_TX\_ENABLE\_MASK**  
 pd.h, 270  
**CCG\_PD\_EXT\_PPS\_STATUS\_SIZE**  
 pd.h, 270  
**CCG\_PD\_EXT\_SRCCAP\_INP\_INDEX**  
 pd.h, 270  
**CCG\_PD\_EXT\_SRCCAP\_INP\_UNCONSTRAINED**  
 pd.h, 270  
**CCG\_PD\_EXT\_SRCCAP\_PDP\_INDEX**  
 pd.h, 270  
**CCG\_PD\_EXT\_SRCCAP\_SIZE**  
 pd.h, 271  
**CCG\_PD\_EXT\_STATUS\_SIZE**  
 pd.h, 271  
**CCG\_PD\_FIX\_SRC\_PDO\_MASK\_REV2**  
 pd.h, 271  
**CCG\_PD\_FIX\_SRC\_PDO\_MASK\_REV3**  
 pd.h, 271  
**CCG\_PD\_FLAG\_CONTRACT\_NEG\_ACTIVE**  
 pd.h, 271  
**CCG\_PD\_FLAG\_EXPLICIT\_CONTRACT**  
 pd.h, 271  
**CCG\_PD\_FLAG\_POWER\_SINK**  
 pd.h, 271  
**CCG\_PD\_FLAG\_SRC\_READY**  
 pd.h, 271  
**CCG\_SILICON\_REV00\_VALUE**  
 hal\_ccgx.h, 322  
**CCG\_STATUS\_CODE\_OFFSET**  
 status.h, 395  
**CY\_VID**  
 pd.h, 271  
**cable\_disc\_en**  
 pd\_port\_config\_t, 96  
**cap\_mismatch**  
 pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
 pd\_do\_t::RDO\_BAT, 110  
 pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
 pd\_do\_t::RDO\_FIXED\_VAR, 113  
 pd\_do\_t::RDO\_GEN\_GVB, 118  
 pd\_do\_t::RDO\_GEN, 116  
**cb**  
 ccg\_timer\_t, 46  
**cb\_fun\_ptr**  
 i2c\_scb\_config\_t, 74  
**cbk**  
 alt\_mode\_info\_t, 26  
**cbl\_disc\_id\_finished**  
 app\_status\_t, 34  
**cbl\_dsc**  
 dpm\_status\_t, 57  
**cbl\_fw\_ver**  
 pd\_do\_t::ACT\_CBL\_VDO, 19  
 pd\_do\_t::PAS\_CBL\_VDO, 79  
 pd\_do\_t::STD\_CBL\_VDO, 125  
**cbl\_hw\_ver**  
 pd\_do\_t::ACT\_CBL\_VDO, 19  
 pd\_do\_t::PAS\_CBL\_VDO, 79  
 pd\_do\_t::STD\_CBL\_VDO, 125  
**cbl\_latency**  
 pd\_do\_t::ACT\_CBL\_VDO, 19  
 pd\_do\_t::PAS\_CBL\_VDO, 79  
 pd\_do\_t::STD\_CBL\_VDO, 125  
**cbl\_mode\_en**  
 dpm\_status\_t, 57  
**cbl\_obj\_pos**  
 alt\_mode\_info\_t, 26  
**cbl\_soft\_reset\_tried**  
 dpm\_status\_t, 57  
**cbl\_sop\_flag**  
 alt\_mode\_reg\_info\_t, 28  
**cbl\_state**  
 dpm\_status\_t, 57  
**cbl\_svid**  
 atch\_tgt\_info\_t, 37  
**cbl\_term**  
 pd.h, 287  
**cbl\_type**  
 dpm\_status\_t, 57  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 144  
**cbl\_vbus\_cur\_t**  
 pd.h, 287  
**cbl\_vdm\_version**  
 dpm\_status\_t, 57  
**cbl\_vdo**  
 atch\_tgt\_info\_t, 37  
 dpm\_status\_t, 57  
**cbl\_wait**  
 dpm\_status\_t, 57  
**cc**  
 cc\_state\_t, 46  
**cc\_live**  
 dpm\_status\_t, 58  
**cc\_old\_status**  
 dpm\_status\_t, 58  
**cc\_rd\_status**  
 dpm\_status\_t, 58  
**cc\_state\_t**, 45  
 cc, 46  
 state, 46  
**cc\_status**  
 dpm\_status\_t, 58  
**ccg\_app\_task\_init**

app.h, 191  
ccg\_get\_si\_revision  
    hal\_ccgx.h, 322  
ccg\_spec\_rev  
    pd\_port\_status\_t::PD\_PORT\_STAT, 104  
ccg\_status\_t  
    status.h, 395  
ccg\_supply\_t  
    pdss\_hal.h, 339  
ccg\_timer\_t, 46  
    cb, 46  
    count, 46  
    id, 47  
    period, 47  
ccgx\_version.h  
    FW\_BASE\_VERSION, 380  
cfg  
    usb\_handle\_t, 136  
chg\_cfg\_params\_t, 47  
    afc\_src\_cap\_cnt, 47  
    afc\_src\_caps, 47  
    apple\_src\_id, 48  
    qc\_src\_type, 48  
    reserved\_0, 48  
    reserved\_1, 48  
    snk\_sel, 48  
    src\_sel, 48  
    table\_len, 48  
chg\_cfg\_tbl\_offset  
    pd\_port\_config\_t, 96  
chunk\_no  
    pd\_extd\_hdr\_t, 89  
    pd\_hdr\_t::PD\_HDR, 90  
chunked  
    pd\_extd\_hdr\_t, 89  
    pd\_hdr\_t::PD\_HDR, 90  
class\_rqt\_cb  
    usb\_config\_t, 134  
clock\_freq  
    i2c\_scb\_config\_t, 74  
cmd  
    pd\_do\_t::STD\_VDM\_HDR, 129  
    pd\_do\_t::USTD\_VDM\_HDR, 149  
    usb\_setup\_pkt\_t, 140  
cmd\_0  
    pd\_do\_t::USTD\_QC\_4\_0\_HDR, 148  
cmd\_1  
    pd\_do\_t::USTD\_QC\_4\_0\_HDR, 148  
cmd\_do  
    dpm\_pd\_cmd\_buf\_t, 53  
cmd\_p  
    dpm\_status\_t, 58  
cmd\_sop  
    dpm\_pd\_cmd\_buf\_t, 53  
cmd\_type  
    pd\_do\_t::STD\_VDM\_HDR, 129  
    pd\_do\_t::USTD\_VDM\_HDR, 149  
comp\_id\_t  
    pdss\_hal.h, 340  
    comp\_tbl\_t, 49  
        alt\_mode\_id, 49  
        svid, 49  
    comp\_tr\_id\_t  
        pdss\_hal.h, 340  
    conn\_orien  
        ridge\_reg\_t::USBPD\_STATUS\_REG, 144  
connect  
    dpm\_status\_t, 58  
contract  
    dpm\_status\_t, 58  
contract\_exist  
    dpm\_status\_t, 58  
    pd\_port\_status\_t::PD\_PORT\_STAT, 104  
contract\_t, 49  
    cur\_pwr, 49  
    max\_volt, 50  
    min\_volt, 50  
count  
    ccg\_timer\_t, 46  
    usb\_i2cm\_t, 138  
crc16  
    utils.h, 407  
cs\_res  
    ocp\_settings\_t, 75  
ctrl  
    usb\_i2cm\_t, 139  
ctrl\_msg\_t  
    pd.h, 288  
cur\_data\_role  
    pd\_port\_status\_t::PD\_PORT\_STAT, 104  
cur\_fb  
    dpm\_status\_t, 58  
cur\_fb\_enabled  
    app\_status\_t, 34  
cur\_port\_role  
    dpm\_status\_t, 59  
cur\_port\_type  
    dpm\_status\_t, 59  
cur\_power\_role  
    pd\_port\_status\_t::PD\_PORT\_STAT, 104  
cur\_pwr  
    contract\_t, 49  
cur\_snk\_max\_min  
    dpm\_status\_t, 59  
cur\_snk\_pdo  
    dpm\_status\_t, 59  
cur\_snk\_pdo\_count  
    dpm\_status\_t, 59  
cur\_src\_pdo  
    dpm\_status\_t, 59  
cur\_src\_pdo\_count  
    dpm\_status\_t, 59  
current\_level  
    pd\_port\_config\_t, 96  
custom\_att\_obj\_pos  
    alt\_mode\_info\_t, 26

cutoff\_volt  
 otp\_settings\_t, 77

DP\_SVID  
 pd.h, 272

DRP\_TOGGLE\_PERIOD  
 pd.h, 272

dat  
 pd\_packet\_extd\_t, 93  
 pd\_packet\_t, 94

dat\_ptr  
 dpm\_pd\_cmd\_buf\_t, 53

data\_0  
 pd\_do\_t::QC\_4\_0\_DATA\_VDO, 109

data\_1  
 pd\_do\_t::QC\_4\_0\_DATA\_VDO, 109

data\_2  
 pd\_do\_t::QC\_4\_0\_DATA\_VDO, 109

data\_3  
 pd\_do\_t::QC\_4\_0\_DATA\_VDO, 110

data\_conn\_pres  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 144

data\_msg\_t  
 pd.h, 288

data\_role  
 alt\_mode\_evt\_t::ALT\_MODE\_EVT, 21  
 alt\_mode\_hw\_evt\_t::ALT\_MODE\_HW\_EVT, 24  
 alt\_mode\_reg\_info\_t, 28  
 pd\_hdr\_t::PD\_HDR, 90  
 pd\_packet\_extd\_t, 93  
 pd\_packet\_t, 94

data\_size  
 pd\_extd\_hdr\_t, 89  
 pd\_hdr\_t::PD\_HDR, 90

db\_support  
 dpm\_status\_t, 59

dbg\_acc\_mode  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

dbg\_alt\_m\_conn  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

dead\_bat  
 dpm\_status\_t, 59

dead\_bat\_support  
 pd\_port\_config\_t, 96

debounce  
 ocp\_settings\_t, 76  
 otp\_settings\_t, 77  
 ovp\_settings\_t, 78  
 scp\_settings\_t, 120  
 uvp\_settings\_t, 150  
 vconn\_ocp\_settings\_t, 152

debounce2  
 ocp\_settings\_t, 76

default\_role  
 pd\_port\_config\_t, 97

default\_sink\_pdo\_mask  
 pd\_port\_config\_t, 97

default\_src\_pdo\_mask  
 pd\_port\_config\_t, 97

dev\_stat  
 usb\_handle\_t, 136

dflt\_data\_pref  
 pd\_port\_status\_t::PD\_PORT\_STAT, 104

dflt\_data\_role  
 pd\_port\_status\_t::PD\_PORT\_STAT, 104

dflt\_port\_role  
 dpm\_status\_t, 60

dflt\_power\_pref  
 pd\_port\_status\_t::PD\_PORT\_STAT, 104

dflt\_power\_role  
 pd\_port\_status\_t::PD\_PORT\_STAT, 104

dfp\_asgmt  
 pd\_do\_t::DP\_CONFIG\_VDO, 50

dfp\_d\_pin  
 pd\_do\_t::STD\_DP\_VDO, 127

dfp\_ufp\_conn  
 pd\_do\_t::DP\_STATUS\_VDO, 51

do\_count  
 vdm\_resp\_t, 154

dock\_cfg\_tbl\_offset  
 pd\_port\_config\_t, 97

dp\_cfg\_vdo  
 pd\_do\_t, 85

dp\_config\_supported  
 pd\_port\_config\_t, 97

dp\_conn  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

dp\_conn\_t  
 dp\_sid.h, 171

dp\_host\_conn  
 ridge\_reg\_t::USBPD\_CMD\_REG, 142

dp\_mode\_trigger  
 pd\_port\_config\_t, 97

dp\_mux\_control  
 pd\_port\_config\_t, 97

dp\_oper  
 pd\_port\_config\_t, 97

dp\_pin\_assign  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

dp\_port\_cap\_t  
 dp\_sid.h, 171

dp\_pref\_mode  
 pd\_port\_config\_t, 97

dp\_role  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

dp\_sid.h  
 dp\_conn\_t, 171  
 dp\_port\_cap\_t, 171  
 dp\_stat\_bm\_t, 171  
 dp\_state\_t, 172  
 reg\_dp\_modes, 172

dp\_snk\_get\_hpd\_state  
 alt\_mode\_hw.h, 158

dp\_stat\_bm\_t  
 dp\_sid.h, 171

dp\_stat\_vdo  
 pd\_do\_t, 86

dp\_state\_t  
    dp\_sid.h, 172  
dpdm\_mux\_cfg\_t  
    pdss\_hal.h, 340  
dpm.h  
    dpm\_clear\_fault\_active, 244  
    dpm\_clear\_hard\_reset\_count, 244  
    dpm\_deepsleep, 245  
    dpm\_disable, 245  
    dpm\_get\_def\_cable\_cap, 245  
    dpm\_get\_evtno\_n\_clear, 246  
    dpm\_get\_info, 246  
    dpm\_get\_mux\_enable\_wait\_period, 246  
    dpm\_get\_pd\_port\_status, 246  
    dpm\_get\_polarity, 247  
    dpm\_get\_sink\_detach\_margin, 247  
    dpm\_get\_sink\_detach\_voltage, 247  
    dpm\_get\_snk\_wait\_cap\_period, 248  
    dpm\_init, 248  
    dpm\_is\_rdo\_valid, 248  
    dpm\_pd\_command, 249  
    dpm\_pd\_command\_ec, 249  
    dpm\_pe\_stop, 250  
    dpm\_prot\_reset, 250  
    dpm\_prot\_reset\_rx, 250  
    dpm\_send\_hard\_reset, 251  
    dpm\_set\_alert, 251  
    dpm\_set\_cf, 252  
    dpm\_set\_chunk\_transfer\_running, 252  
    dpm\_set\_fault\_active, 252  
    dpm\_sleep, 253  
    dpm\_start, 253  
    dpm\_stop, 253  
    dpm\_task, 253  
    dpm\_typec\_command, 254  
    dpm\_typec\_deassert\_rp\_rd, 254  
    dpm\_update\_def\_cable\_cap, 255  
    dpm\_update\_ext\_src\_cap, 255  
    dpm\_update\_frs\_enable, 255  
    dpm\_update\_mux\_enable\_wait\_period, 256  
    dpm\_update\_port\_config, 256  
    dpm\_update\_port\_status, 257  
    dpm\_update\_snk\_cap, 257  
    dpm\_update\_snk\_cap\_mask, 257  
    dpm\_update\_snk\_max\_min, 258  
    dpm\_update\_snk\_wait\_cap\_period, 258  
    dpm\_update\_src\_cap, 259  
    dpm\_update\_src\_cap\_mask, 259  
    dpm\_update\_swap\_response, 259  
    dpm\_wakeup, 260  
dpm\_clear\_fault\_active  
    dpm.h, 244  
dpm\_clear\_hard\_reset\_count  
    dpm.h, 244  
dpm\_cmd\_buf  
    dpm\_status\_t, 60  
dpm\_deepsleep  
    dpm.h, 245  
dpm\_disable  
    dpm.h, 245  
dpm\_enabled  
    dpm\_status\_t, 60  
dpm\_err\_info  
    dpm\_status\_t, 60  
dpm\_get\_def\_cable\_cap  
    dpm.h, 245  
dpm\_get\_evtno\_n\_clear  
    dpm.h, 246  
dpm\_get\_info  
    dpm.h, 246  
dpm\_get\_mux\_enable\_wait\_period  
    dpm.h, 246  
dpm\_get\_pd\_port\_status  
    dpm.h, 246  
dpm\_get\_polarity  
    dpm.h, 247  
dpm\_get\_sink\_detach\_margin  
    dpm.h, 247  
dpm\_get\_sink\_detach\_voltage  
    dpm.h, 247  
dpm\_get\_snk\_wait\_cap\_period  
    dpm.h, 248  
dpm\_init  
    dpm.h, 248  
    dpm\_status\_t, 60  
dpm\_is\_rdo\_valid  
    dpm.h, 248  
dpm\_pd\_cbk  
    dpm\_status\_t, 60  
dpm\_pd\_cmd  
    dpm\_status\_t, 60  
dpm\_pd\_cmd\_active  
    dpm\_status\_t, 60  
dpm\_pd\_cmd\_buf\_t, 52  
    cmd\_do, 53  
    cmd\_sop, 53  
    dat\_ptr, 53  
    extd\_hdr, 53  
    extd\_type, 53  
    no\_of\_cmd\_do, 53  
    timeout, 53  
dpm\_pd\_cmd\_cbk\_t  
    pd.h, 283  
dpm\_pd\_cmd\_t  
    pd.h, 289  
dpm\_pd\_command  
    dpm.h, 249  
dpm\_pd\_command\_ec  
    dpm.h, 249  
dpm\_pe\_stop  
    dpm.h, 250  
dpm\_prot\_reset  
    dpm.h, 250  
dpm\_prot\_reset\_rx  
    dpm.h, 250  
dpm\_safe\_disable

dpm\_status\_t, 60  
 dpm\_send\_hard\_reset  
     dpm.h, 251  
 dpm\_set\_alert  
     dpm.h, 251  
 dpm\_set\_cf  
     dpm.h, 252  
 dpm\_set\_chunk\_transfer\_running  
     dpm.h, 252  
 dpm\_set\_fault\_active  
     dpm.h, 252  
 dpm\_sleep  
     dpm.h, 253  
 dpm\_start  
     dpm.h, 253  
 dpm\_status\_t, 54  
     alert, 56  
     app\_cbk, 56  
     attach, 56  
     attached\_dev, 56  
     bist\_cm2\_enabled, 56  
     bootup, 57  
     cbl\_dsc, 57  
     cbl\_mode\_en, 57  
     cbl\_soft\_reset\_tried, 57  
     cbl\_state, 57  
     cbl\_type, 57  
     cbl\_vdm\_version, 57  
     cbl\_vdo, 57  
     cbl\_wait, 57  
     cc\_live, 58  
     cc\_old\_status, 58  
     cc\_rd\_status, 58  
     cc\_status, 58  
     cmd\_p, 58  
     connect, 58  
     contract, 58  
     contract\_exist, 58  
     cur\_fb, 58  
     cur\_port\_role, 59  
     cur\_port\_type, 59  
     cur\_snk\_max\_min, 59  
     cur\_snk\_pdo, 59  
     cur\_snk\_pdo\_count, 59  
     cur\_src\_pdo, 59  
     cur\_src\_pdo\_count, 59  
     db\_support, 59  
     dead\_bat, 59  
     dflt\_port\_role, 60  
     dpm\_cmd\_buf, 60  
     dpm\_enabled, 60  
     dpm\_err\_info, 60  
     dpm\_init, 60  
     dpm\_pd\_cbk, 60  
     dpm\_pd\_cmd, 60  
     dpm\_pd\_cmd\_active, 60  
     dpm\_safe\_disable, 60  
     dpm\_typec\_cbk, 61  
     dpm\_typec\_cmd, 61  
     dpm\_typec\_cmd\_active, 61  
     drp\_period, 61  
     emca\_present, 61  
     err\_recov, 61  
     ext\_src\_cap, 61  
     fault\_active, 61  
     fr\_rx\_disabled, 61  
     fr\_rx\_en\_live, 62  
     fr\_tx\_disabled, 62  
     fr\_tx\_en\_live, 62  
     frs\_enable, 62  
     is\_snk\_bat, 62  
     is\_src\_bat, 62  
     non\_intr\_response, 62  
     padval, 62  
     pd\_connected, 62  
     pd\_disabled, 63  
     pd\_support, 63  
     pe\_evt, 63  
     pe\_fsm\_state, 63  
     polarity, 63  
     port\_disable, 63  
     port\_role, 63  
     port\_status, 63  
     pps\_status, 63  
     ra\_present, 64  
     reserved\_3, 64  
     rev\_pol, 64  
     role\_at\_connect, 64  
     rp\_supported, 64  
     skip\_scan, 64  
     snk\_cur\_level, 64  
     snk\_max\_min, 64  
     snk\_pdo, 65  
     snk\_pdo\_count, 65  
     snk\_pdo\_mask, 65  
     snk\_period, 65  
     snk\_rdo, 65  
     snk\_rp\_detach\_en, 65  
     snk\_sel\_pdo, 65  
     snk\_usb\_comm\_en, 65  
     snk\_usb\_susp\_en, 65  
     spec\_rev\_cbl, 66  
     spec\_rev\_peer, 66  
     spec\_rev\_sop\_live, 66  
     spec\_rev\_sop\_prime\_live, 66  
     src\_cap\_p, 66  
     src\_cur\_level, 66  
     src\_cur\_level\_live, 66  
     src\_cur\_rdo, 66  
     src\_last\_rdo, 66  
     src\_pdo, 67  
     src\_pdo\_count, 67  
     src\_pdo\_mask, 67  
     src\_period, 67  
     src\_rdo, 67  
     src\_sel\_pdo, 67

swap\_response, 67  
toggle, 67  
try\_src\_snk, 68  
typec\_evt, 68  
typec\_fsm\_state, 68  
unchunk\_sup\_live, 68  
unchunk\_sup\_peer, 68  
vconn\_logical, 68  
vconn\_retain, 68  
dpm\_stop  
    dpm.h, 253  
dpm\_task  
    dpm.h, 253  
dpm\_typec\_cbk  
    dpm\_status\_t, 61  
dpm\_typec\_cmd  
    dpm\_status\_t, 61  
dpm\_typec\_cmd\_active  
    dpm\_status\_t, 61  
dpm\_typec\_cmd\_cbk\_t  
    pd.h, 283  
dpm\_typec\_cmd\_resp\_t  
    pd.h, 289  
dpm\_typec\_cmd\_t  
    pd.h, 290  
dpm\_typec\_command  
    dpm.h, 254  
dpm\_typec\_deassert\_rp\_rd  
    dpm.h, 254  
dpm\_update\_def\_cable\_cap  
    dpm.h, 255  
dpm\_update\_ext\_src\_cap  
    dpm.h, 255  
dpm\_update\_frs\_enable  
    dpm.h, 255  
dpm\_update\_mux\_enable\_wait\_period  
    dpm.h, 256  
dpm\_update\_port\_config  
    dpm.h, 256  
dpm\_update\_port\_status  
    dpm.h, 257  
dpm\_update\_snk\_cap  
    dpm.h, 257  
dpm\_update\_snk\_cap\_mask  
    dpm.h, 257  
dpm\_update\_snk\_max\_min  
    dpm.h, 258  
dpm\_update\_snk\_wait\_cap\_period  
    dpm.h, 258  
dpm\_update\_src\_cap  
    dpm.h, 259  
dpm\_update\_src\_cap\_mask  
    dpm.h, 259  
dpm\_update\_swap\_response  
    dpm.h, 259  
dpm\_wakeup  
    dpm.h, 260  
dr\_swap  
    pd\_do\_t::FIXED\_SNK, 69  
    pd\_do\_t::FIXED\_SRC, 70  
drp\_period  
    dpm\_status\_t, 61  
drp\_toggle\_en  
    pd\_port\_config\_t, 98  
dual\_role\_power  
    pd\_do\_t::FIXED\_SNK, 69  
    pd\_do\_t::FIXED\_SRC, 70  
dummy  
    pd\_power\_status\_t, 107  
emca\_present  
    dpm\_status\_t, 61  
    pd\_port\_status\_t::PD\_PORT\_STAT, 104  
emca\_spec\_rev  
    pd\_port\_status\_t::PD\_PORT\_STAT, 105  
en  
    pd\_do\_t::DP\_STATUS\_VDO, 51  
enable\_vdm\_task\_mngr  
    vdm\_task\_mngr.h, 179  
enabled  
    usb\_ep\_handle\_t, 135  
ep0\_buffer  
    bb\_handle\_t, 41  
    usb\_handle\_t, 136  
ep0\_last  
    usb\_handle\_t, 136  
ep0\_length  
    usb\_handle\_t, 137  
ep0\_state  
    usb\_handle\_t, 137  
ep0\_toggle  
    usb\_handle\_t, 137  
ep0\_xfer\_cb  
    usb\_handle\_t, 137  
ep0\_zlp\_rqd  
    usb\_handle\_t, 137  
ep\_handle  
    usb\_handle\_t, 137  
err\_recov  
    dpm\_status\_t, 61  
err\_recovery\_en  
    pd\_port\_config\_t, 98  
eval\_app\_alt\_hw\_cmd  
    alt\_mode\_hw.h, 159  
eval\_app\_alt\_mode\_cmd  
    alt\_modes\_mngr.h, 166  
eval\_app\_cmd  
    alt\_mode\_info\_t, 26  
eval\_dr\_swap  
    app\_cbk\_t, 30  
    swap.h, 213  
eval\_fr\_swap  
    app\_cbk\_t, 30  
eval\_hpd\_cmd  
    alt\_mode\_hw.h, 159  
eval\_mux\_cmd  
    alt\_mode\_hw.h, 159

eval\_pr\_swap  
     app\_cbk\_t, 30  
     swap.h, 214  
 eval\_rdo  
     app\_cbk\_t, 30  
     pdo.h, 204  
 eval\_rec\_vdm  
     alt\_modes\_mngr.h, 166  
 eval\_src\_cap  
     app\_cbk\_t, 30  
     pdo.h, 205  
 eval\_vconn\_swap  
     app\_cbk\_t, 30  
     swap.h, 214  
 eval\_vdm  
     app\_cbk\_t, 30  
     vdm.h, 226  
 event\_cb  
     usb\_config\_t, 134  
 event\_flags  
     pd\_power\_status\_t, 107  
 evt\_data  
     alt\_mode\_evt\_t::ALT\_MODE\_EVT\_DATA, 22  
     alt\_mode\_hw\_evt\_t::ALT\_MODE\_HW\_EVT, 24  
 evt\_type  
     alt\_mode\_evt\_t::ALT\_MODE\_EVT\_DATA, 22  
 exit  
     pd\_do\_t::DP\_STATUS\_VDO, 51  
 extPowered  
     pd\_do\_t::FIXED\_SNK, 69  
     pd\_do\_t::FIXED\_SRC, 71  
 extSrcCap  
     dpm\_status\_t, 61  
 extSrcCapLength  
     pd\_port\_config\_t, 98  
 extSrcCapOffset  
     pd\_port\_config\_t, 98  
 extd  
     pd\_extd\_hdr\_t, 89  
     pd\_hdr\_t::PD\_HDR, 91  
 extdHdr  
     dpm\_pd\_cmd\_buf\_t, 53  
 extdMsg\_t  
     pd.h, 290  
 extdType  
     dpm\_pd\_cmd\_buf\_t, 53  
 FRS\_TX\_SWAP\_CTRL1\_DFLT\_VAL  
     pdss\_hal.h, 335  
 FW\_BASE\_VERSION  
     ccgx\_version.h, 380  
 failStatus\_t  
     alt\_modes\_mngr.h, 164  
 fallback\_rqst\_cb  
     usb\_config\_t, 134  
 faultActive  
     dpm\_status\_t, 61  
 faultStatus  
     app\_status\_t, 34  
 fb\_ctrl\_r1  
     pwr\_params\_t, 108  
 fb\_ctrl\_r2  
     pwr\_params\_t, 108  
 fb\_type  
     pwr\_params\_t, 108  
 filter\_edge\_detect\_cfg\_t  
     pdss\_hal.h, 341  
 filter\_id\_t  
     pdss\_hal.h, 341  
 fixed\_snk  
     pd\_do\_t, 86  
 fixed\_src  
     pd\_do\_t, 86  
 flash.h  
     flash\_access\_get\_status, 383  
     flash\_app\_priority\_t, 382  
     flash\_cbk\_t, 382  
     flash\_enter\_mode, 383  
     flash\_interface\_t, 382  
     flash\_row\_clear, 384  
     flash\_row\_read, 384  
     flash\_row\_write, 384  
     flash\_set\_access\_limits, 385  
     flash\_set\_app\_priority, 386  
     flash\_write\_status\_t, 383  
 flash\_access\_get\_status  
     flash.h, 383  
 flash\_app\_priority\_t  
     flash.h, 382  
 flash\_cbk\_t  
     flash.h, 382  
 flash\_checksum  
     pd\_config\_t, 81  
 flash\_enter\_mode  
     flash.h, 383  
 flash\_interface\_t  
     flash.h, 382  
 flash\_row\_clear  
     flash.h, 384  
 flash\_row\_read  
     flash.h, 384  
 flash\_row\_write  
     flash.h, 384  
 flash\_set\_access\_limits  
     flash.h, 385  
 flash\_set\_app\_priority  
     flash.h, 386  
 flash\_write\_status\_t  
     flash.h, 383  
 flashing\_mode  
     bb\_handle\_t, 41  
     pd\_config\_t, 81  
 flashing\_vid  
     pd\_config\_t, 81  
 force\_lsx  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 145  
 form\_alt\_mode\_event

alt\_modes\_mngr.h, 167  
fr\_rx\_disabled  
    dpm\_status\_t, 61  
fr\_rx\_en\_live  
    dpm\_status\_t, 62  
fr\_swap  
    pd\_do\_t::FIXED\_SNK, 69  
fr\_swap\_supp\_t  
    pd.h, 291  
fr\_tx\_disabled  
    dpm\_status\_t, 62  
fr\_tx\_en\_live  
    dpm\_status\_t, 62  
frs\_enable  
    dpm\_status\_t, 62  
    pd\_port\_config\_t, 98  
fw1\_invalid  
    fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
fw2\_invalid  
    fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
fw\_checksum  
    sys\_fw\_metadata\_t, 132  
fw\_entry  
    sys\_fw\_metadata\_t, 133  
fw\_img\_status\_t, 72  
    status, 72  
    val, 72  
fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
    boot\_mode\_request, 73  
    fw1\_invalid, 73  
    fw2\_invalid, 73  
    reserved, 73  
    reserved1, 73  
fw\_size  
    sys\_fw\_metadata\_t, 133  
fw\_version  
    sys\_fw\_metadata\_t, 133

GET\_DR\_SWAP\_RESP  
    pd.h, 272  
GET\_PR\_SWAP\_RESP  
    pd.h, 272  
GET\_VCONN\_SWAP\_RESP  
    pd.h, 272  
GIVE\_BACK\_MASK  
    pd.h, 272  
get\_alt\_mode\_numb  
    alt\_modes\_mngr.h, 167  
get\_boot\_mode\_reason  
    boot.h, 380  
get\_dscr\_cb  
    usb\_config\_t, 134  
get\_hpi\_sof\_reset\_timer\_id  
    hpi.h, 231  
get\_hpi\_soft\_reset\_delay  
    hpi.h, 231  
get\_modes\_vdo\_info  
    vdm.h, 226  
get\_mux\_state

    alt\_mode\_hw.h, 160  
get\_pd\_config  
    pd.h, 302  
get\_pd\_port\_config  
    pd.h, 302  
get\_pe\_state\_buf  
    pd\_policy\_engine.h, 308  
get\_silicon\_revision  
    system.h, 397  
get\_spec\_rev\_determined  
    pd\_policy\_engine.h, 308  
get\_vdm\_buff  
    alt\_modes\_mngr.h, 168  
give\_back\_flag  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
    pd\_do\_t::RDO\_BAT, 110  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
    pd\_do\_t::RDO\_FIXED\_VAR, 113  
    pd\_do\_t::RDO\_GEN\_GVB, 118  
    pd\_do\_t::RDO\_GEN, 116  
gpio.h  
    gpio\_clear\_intr, 391  
    gpio\_dm\_t, 387  
    gpio\_get\_intr, 391  
    gpio\_hsiom\_set\_config, 391  
    gpio\_int\_set\_config, 392  
    gpio\_intr\_t, 388  
    gpio\_port\_pin\_t, 388  
    gpio\_read\_value, 392  
    gpio\_set\_drv\_mode, 393  
    gpio\_set\_lvttl\_mode, 393  
    gpio\_set\_value, 393  
    hsiom\_mode\_t, 390  
    hsiom\_set\_config, 394  
gpio\_clear\_intr  
    gpio.h, 391  
gpio\_dm\_t  
    gpio.h, 387  
gpio\_get\_intr  
    gpio.h, 391  
gpio\_hsiom\_set\_config  
    gpio.h, 391  
gpio\_int\_set\_config  
    gpio.h, 392  
gpio\_intr\_t  
    gpio.h, 388  
gpio\_port\_pin\_t  
    gpio.h, 388  
gpio\_read\_value  
    gpio.h, 392  
gpio\_set\_drv\_mode  
    gpio.h, 393  
gpio\_set\_lvttl\_mode  
    gpio.h, 393  
gpio\_set\_value  
    gpio.h, 393

HEADER\_INFO\_CFG  
    pdss\_hal.h, 335

HPD\_RX\_ACTIVITY\_TIMER\_PERIOD\_MAX  
     pd.h, 272  
 HPD\_RX\_ACTIVITY\_TIMER\_PERIOD\_MIN  
     pd.h, 272  
 hal\_ccgx.h  
     CCG\_SILICON\_REV00\_VALUE, 322  
     ccg\_get\_si\_revision, 322  
     system\_connect\_ovp\_trip, 323  
     system\_disconnect\_ovp\_trip, 323  
     system\_init, 323  
     system\_vbus\_ocp\_dis, 323  
     system\_vbus\_ocp\_en, 324  
     system\_vbus\_scp\_dis, 324  
     system\_vbus\_scp\_en, 324  
     vbus\_ocp\_handler, 325  
     vbus\_scp\_handler, 325  
 hdr  
     pd\_hdr\_t, 92  
     pd\_packet\_extd\_t, 93  
     pd\_packet\_t, 94  
 hid\_report\_id\_t  
     usb\_hid.h, 215  
 hid\_rqt\_cmd\_t  
     usb\_hid.h, 216  
 high\_cap  
     pd\_do\_t::FIXED\_SNK, 69  
 hpd.h  
     hpd\_deinit, 327  
     hpd\_event\_type\_t, 327  
     hpd\_receive\_get\_status, 327  
     hpd\_receive\_init, 328  
     hpd\_rx\_sleep\_entry, 328  
     hpd\_rx\_wakeup, 328  
     hpd\_sleep\_entry, 329  
     hpd\_transmit\_init, 329  
     hpd\_transmit\_sendevt, 329  
     hpd\_wakeup, 330  
     is\_hpd\_rx\_state\_idle, 330  
 hpd\_deinit  
     hpd.h, 327  
 hpd\_event\_type\_t  
     hpd.h, 327  
 hpd\_irq  
     pd\_do\_t::DP\_STATUS\_VDO, 52  
     ridge\_reg\_t::USBPD\_CMD\_REG, 142  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 145  
 hpd\_lv1  
     ridge\_reg\_t::USBPD\_CMD\_REG, 142  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 145  
 hpd\_receive\_get\_status  
     hpd.h, 327  
 hpd\_receive\_init  
     hpd.h, 328  
 hpd\_rx\_sleep\_entry  
     hpd.h, 328  
 hpd\_rx\_wakeup  
     hpd.h, 328  
 hpd\_sleep\_entry

    hpd.h, 329  
 hpd\_state  
     pd\_do\_t::DP\_STATUS\_VDO, 52  
 hpd\_transmit\_init  
     hpd.h, 329  
 hpd\_transmit\_sendevt  
     hpd.h, 329  
 hpd\_wakeup  
     hpd.h, 330  
 hpi.h  
     get\_hpi\_sof\_reset\_timer\_id, 231  
     get\_hpi\_soft\_reset\_delay, 231  
     hpi\_boot\_prio\_conf\_t, 230  
     hpi\_deinit, 232  
     hpi\_get\_port\_enable, 232  
     hpi\_get\_sys\_pwr\_state, 232  
     hpi\_init, 232  
     hpi\_init\_userdef\_regs, 233  
     hpi\_is\_accessed, 233  
     hpi\_is\_ec\_ready, 233  
     hpi\_is\_extd\_msg\_ec\_ctrl\_enabled, 234  
     hpi\_is\_vdm\_ec\_ctrl\_enabled, 234  
     hpi\_pd\_event\_handler, 234  
     hpi\_reg\_enqueue\_event, 235  
     hpi\_reg\_part\_t, 230  
     hpi\_reg\_section\_t, 231  
     hpi\_send\_fw\_ready\_event, 235  
     hpi\_send\_hw\_error\_event, 235  
     hpi\_set\_boot\_priority\_conf, 236  
     hpi\_set\_fixed\_slave\_address, 236  
     hpi\_set\_flash\_params, 236  
     hpi\_set\_mode\_regs, 237  
     hpi\_set\_no\_boot\_mode, 237  
     hpi\_set\_port\_event\_mask, 237  
     hpi\_set\_reserved\_reg\_35, 238  
     hpi\_set\_reserved\_reg\_37, 238  
     hpi\_set\_reset\_count, 238  
     hpi\_set\_userdef\_write\_handler, 240  
     hpi\_sleep, 240  
     hpi\_sleep\_allowed, 240  
     hpi\_task, 240  
     hpi\_update\_fw\_locations, 241  
     hpi\_update\_pdo\_change, 241  
     hpi\_update\_versions, 241  
     hpi\_write\_cb\_t, 230  
     hpid\_get\_ec\_active\_modes, 242  
     update\_hpi\_sof\_reset\_timer\_id, 242  
     update\_hpi\_soft\_reset\_delay, 242  
 hpi\_boot\_prio\_conf\_t  
     hpi.h, 230  
 hpi\_deinit  
     hpi.h, 232  
 hpi\_get\_port\_enable  
     hpi.h, 232  
 hpi\_get\_sys\_pwr\_state  
     hpi.h, 232  
 hpi\_init  
     hpi.h, 232

hpi\_init\_userdef\_regs  
    `hpi.h`, 233

hpi\_is\_accessed  
    `hpi.h`, 233

hpi\_is\_ec\_ready  
    `hpi.h`, 233

hpi\_is\_extd\_msg\_ec\_ctrl\_enabled  
    `hpi.h`, 234

hpi\_is\_vdm\_ec\_ctrl\_enabled  
    `hpi.h`, 234

hpi\_pd\_event\_handler  
    `hpi.h`, 234

hpi\_reg\_enqueue\_event  
    `hpi.h`, 235

hpi\_reg\_part\_t  
    `hpi.h`, 230

hpi\_reg\_section\_t  
    `hpi.h`, 231

hpi\_send\_fw\_ready\_event  
    `hpi.h`, 235

hpi\_send\_hw\_error\_event  
    `hpi.h`, 235

hpi\_set\_boot\_priority\_conf  
    `hpi.h`, 236

hpi\_set\_fixed\_slave\_address  
    `hpi.h`, 236

hpi\_set\_flash\_params  
    `hpi.h`, 236

hpi\_set\_mode\_regs  
    `hpi.h`, 237

hpi\_set\_no\_boot\_mode  
    `hpi.h`, 237

hpi\_set\_port\_event\_mask  
    `hpi.h`, 237

hpi\_set\_reserved\_reg\_35  
    `hpi.h`, 238

hpi\_set\_reserved\_reg\_37  
    `hpi.h`, 238

hpi\_set\_reset\_count  
    `hpi.h`, 238

hpi\_set\_userdef\_write\_handler  
    `hpi.h`, 240

hpi\_sleep  
    `hpi.h`, 240

hpi\_sleep\_allowed  
    `hpi.h`, 240

hpi\_task  
    `hpi.h`, 240

hpi\_update\_fw\_locations  
    `hpi.h`, 241

hpi\_update\_pdo\_change  
    `hpi.h`, 241

hpi\_update\_versions  
    `hpi.h`, 241

hpi\_write\_cb\_t  
    `hpi.h`, 230

hpid\_get\_ec\_active\_modes  
    `hpi.h`, 242

hpiss/hpi.h, 228

hsiom\_mode\_t  
    `gpio.h`, 390

hsiom\_set\_config  
    `gpio.h`, 394

hw\_evt  
    `alt_mode_hw_evt_t`, 24

hw\_type  
    `alt_mode_hw_evt_t::ALT_MODE_HW_EVT`, 24

I2C\_BLOCK\_COUNT  
    `i2c.h`, 371

i2c.h  
    I2C\_BLOCK\_COUNT, 371

i2c\_cb\_cmd\_t, 371

i2c\_reset, 373

i2c\_scb\_clock\_freq\_t, 372

i2c\_scb\_deinit, 373

i2c\_scb\_enable\_wakeup, 373

i2c\_scb\_init, 374

i2c\_scb\_is\_idle, 375

i2c\_scb\_mode\_t, 372

i2c\_scb\_state\_t, 372

i2c\_scb\_write, 375

i2c\_slave\_ack\_ctrl, 375

i2c\_cb\_cmd\_t  
    `i2c.h`, 371

i2c\_int\_ack  
    `ridge_reg_t::USBPD_CMD_REG`, 142

i2c\_reset  
    `i2c.h`, 373

i2c\_scb\_clock\_freq\_t  
    `i2c.h`, 372

i2c\_scb\_config\_t, 73

    buf\_size, 74

    buffer, 74

    cb\_fun\_ptr, 74

    clock\_freq, 74

    i2c\_state, 74

    i2c\_write\_count, 74

    mode, 75

    slave\_address, 75

    slave\_mask, 75

i2c\_scb\_deinit  
    `i2c.h`, 373

i2c\_scb\_enable\_wakeup  
    `i2c.h`, 373

i2c\_scb\_init  
    `i2c.h`, 374

i2c\_scb\_is\_idle  
    `i2c.h`, 375

i2c\_scb\_mode\_t  
    `i2c.h`, 372

i2c\_scb\_state\_t  
    `i2c.h`, 372

i2c\_scb\_write  
    `i2c.h`, 375

i2c\_slave\_ack\_ctrl  
    `i2c.h`, 375

i2c\_state  
 i2c\_scb\_config\_t, 74

i2c\_write\_count  
 i2c\_scb\_config\_t, 74

I\_1P5A  
 pd.h, 273

I\_1A  
 pd.h, 273

I\_2A  
 pd.h, 273

I\_3A  
 pd.h, 273

I\_5A  
 pd.h, 273

ID\_HEADER\_IDX  
 pd.h, 273

ISAFE\_0A  
 pd.h, 273

ISAFE\_DEF  
 pd.h, 273

id  
 ccg\_timer\_t, 47

id\_vdm\_length  
 pd\_port\_config\_t, 98

id\_vdm\_offset  
 pd\_port\_config\_t, 98

index  
 usb\_setup\_pkt\_t, 140

int\_event  
 usb\_handle\_t, 137

intel\_ridge.h  
 ridge\_eval\_cmd, 174  
 ridge\_set\_ctrl\_change\_cb, 174  
 ridge\_set\_mux, 174  
 tr\_hpd\_deinit, 175  
 tr\_hpd\_init, 175  
 tr\_hpd\_sendevt, 175  
 tr\_is\_hpd\_change, 176

intel\_vid.h  
 reg\_intel\_modes, 177  
 tbt\_state\_t, 177

interrupt\_ack  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 145

intl\_temperature  
 pd\_power\_status\_t, 107

irq\_ack  
 ridge\_reg\_t::USBPD\_CMD\_REG, 142  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 146

is\_active  
 alt\_mode\_info\_t, 26

is\_alt\_mode\_mngr\_idle  
 alt\_modes\_mngr.h, 168

is\_hpd\_rx\_state\_idle  
 hpd.h, 330

is\_mux\_busy  
 app\_status\_t, 34

is\_out  
 usb\_ep\_handle\_t, 135

is\_snk\_bat  
 dpm\_status\_t, 62  
 pd\_port\_config\_t, 98

is\_src\_bat  
 dpm\_status\_t, 62  
 pd\_port\_config\_t, 98

is\_svid\_supported  
 alt\_modes\_mngr.h, 168

is\_vbus\_on  
 app\_status\_t, 34

is\_vconn\_on  
 app\_status\_t, 35

is\_vdm\_pending  
 app\_status\_t, 35

is\_vdm\_task\_idle  
 vdm\_task\_mngr.h, 179

ld\_sw\_ctrl  
 app\_status\_t, 35

len  
 pd\_hdr\_t::PD\_HDR, 91  
 pd\_packet\_extd\_t, 93  
 pd\_packet\_t, 94

length  
 usb\_i2cm\_t, 139  
 usb\_setup\_pkt\_t, 140

lscsa\_app\_config\_t  
 pdss\_hal.h, 341

MAKE\_DWORD  
 utils.h, 407

MAX\_CBL\_DSC\_ID\_COUNT  
 pd.h, 273

MAX\_EXTD\_MSG\_LEGACY\_LEN  
 pd.h, 274

MAX\_EXTD\_PKT\_SIZE  
 pd.h, 274

MAX\_EXTD\_PKT\_WORDS  
 pd.h, 274

MAX\_HARD\_RESET\_COUNT  
 pd.h, 274

MAX\_MESSAGE\_ID  
 pd.h, 274

MAX\_NO\_OF\_DO  
 pd.h, 274

MAX\_NO\_OF\_PDO  
 pd.h, 274

MAX\_NO\_OF\_VDO  
 pd.h, 274

MAX\_SOP\_TYPES  
 pd.h, 274

MAX\_SRC\_CAP\_COUNT  
 pd.h, 275

max\_cur\_power  
 pd\_do\_t::SRC\_GEN, 121

max\_current  
 pd\_do\_t::FIXED\_SRC, 71  
 pd\_do\_t::VAR\_SRC, 151

max\_op\_current

pd\_do\_t::RDO\_FIXED\_VAR, 113  
max\_op\_power  
    pd\_do\_t::RDO\_BAT, 110  
max\_power  
    pd\_do\_t::BAT\_SRC, 40  
max\_power\_cur  
    pd\_do\_t::RDO\_GEN\_GVB, 118  
max\_vbus\_VOLT  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::PAS\_CBL\_VDO, 79  
max\_VOLT  
    contract\_t, 50  
max\_voltage  
    pd\_do\_t::BAT\_SNK, 39  
    pd\_do\_t::BAT\_SRC, 40  
    pd\_do\_t::SRC\_GEN, 121  
    pd\_do\_t::VAR\_SNK, 151  
    pd\_do\_t::VAR\_SRC, 151  
mem\_calculate\_byte\_checksum  
    utils.h, 408  
mem\_calculate\_dword\_checksum  
    utils.h, 408  
mem\_calculate\_word\_checksum  
    utils.h, 408  
mem\_copy\_word  
    utils.h, 409  
metadata\_valid  
    sys\_fw\_metadata\_t, 133  
mfg\_info\_length  
    pd\_config\_t, 81  
mfg\_info\_offset  
    pd\_config\_t, 82  
min\_max\_power\_cur  
    pd\_do\_t::RDO\_GEN, 117  
min\_op\_current  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
min\_op\_power  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
min\_state  
    pd\_port\_status\_t::PD\_PORT\_STAT, 105  
min\_VOLT  
    contract\_t, 50  
min\_voltage  
    pd\_do\_t::BAT\_SNK, 39  
    pd\_do\_t::BAT\_SRC, 40  
    pd\_do\_t::SRC\_GEN, 121  
    pd\_do\_t::VAR\_SNK, 151  
    pd\_do\_t::VAR\_SRC, 152  
mod\_support  
    pd\_do\_t::STD\_VDM\_ID\_HDR, 131  
mode  
    i2c\_scb\_config\_t, 75  
    pd\_do\_t::BIST\_DO, 45  
mode\_state  
    alt\_mode\_info\_t, 26  
mode\_vdm\_length  
    pd\_port\_config\_t, 99  
mode\_vdm\_offset

pd\_port\_config\_t, 99  
msg  
    pd\_packet\_extd\_t, 93  
    pd\_packet\_t, 94  
msg\_id  
    pd\_hdr\_t::PD\_HDR, 91  
msg\_type  
    pd\_hdr\_t::PD\_HDR, 91  
mult\_fun  
    pd\_do\_t::DP\_STATUS\_VDO, 52  
mux\_ctrl\_init  
    app.h, 192  
mux\_ctrl\_set\_cfg  
    app.h, 192  
mux\_poll\_cbk  
    app\_status\_t, 35  
mux\_poll\_fnc\_cbk\_t  
    app.h, 184  
mux\_poll\_status\_t  
    alt\_mode\_hw.h, 156  
mux\_select\_t  
    alt\_mode\_hw.h, 156  
no\_of\_cmd\_do  
    dpm\_pd\_cmd\_buf\_t, 53  
no\_resp  
    vdm\_resp\_t, 154  
no\_usb\_suspend  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
    pd\_do\_t::RDO\_BAT, 110  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
    pd\_do\_t::RDO\_FIXED\_VAR, 114  
    pd\_do\_t::RDO\_GEN\_GVB, 118  
    pd\_do\_t::RDO\_GEN, 117  
non\_intr\_response  
    dpm\_status\_t, 62  
num\_alt\_modes  
    bb\_handle\_t, 41  
obj\_pos  
    alt\_mode\_info\_t, 26  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
    pd\_do\_t::RDO\_BAT, 111  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
    pd\_do\_t::RDO\_FIXED\_VAR, 114  
    pd\_do\_t::RDO\_GEN\_GVB, 118  
    pd\_do\_t::RDO\_GEN, 117  
    pd\_do\_t::STD\_VDM\_HDR, 130  
ocp\_settings\_t, 75  
    cs\_res, 75  
    debounce, 76  
    debounce2, 76  
    retry\_cnt, 76  
    sense\_res, 76  
    table\_len, 76  
    threshold, 76  
    threshold2, 76  
ocp\_tbl\_offset  
    pd\_port\_config\_t, 99

op\_current  
 pd\_do\_t::FIXED\_SNK, 69  
 pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
 pd\_do\_t::RDO\_FIXED\_VAR, 114  
 pd\_do\_t::VAR\_SNK, 151

op\_power  
 pd\_do\_t::BAT\_SNK, 39  
 pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
 pd\_do\_t::RDO\_BAT, 111

op\_power\_cur  
 pd\_do\_t::RDO\_GEN\_GVB, 118  
 pd\_do\_t::RDO\_GEN, 117

otp\_settings\_t, 76  
 cutoff\_volt, 77  
 debounce, 77  
 reserved\_0, 77  
 restart\_volt, 77  
 table\_len, 77  
 therm\_type, 77

otp\_tbl\_offset  
 pd\_port\_config\_t, 99

ovc\_indn  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 146

ovp\_settings\_t, 78  
 debounce, 78  
 retry\_cnt, 78  
 table\_len, 78  
 threshold, 78

ovp\_tbl\_offset  
 pd\_port\_config\_t, 99

PD\_ADC\_CB\_T  
 pdss\_hal.h, 338

PD\_ADC\_ID\_T  
 pdss\_hal.h, 342

PD\_ADC\_INPUT\_T  
 pdss\_hal.h, 342

PD\_ADC\_INT\_T  
 pdss\_hal.h, 342

PD\_BIST\_CONT\_MODE\_TIMER\_PERIOD  
 pd.h, 275

PD\_CBL\_DELAY\_TIMER\_PERIOD  
 pd.h, 275

PD\_CBL\_DSC\_ID\_START\_TIMER\_PERIOD  
 pd.h, 275

PD\_CBL\_DSC\_ID\_TIMER\_PERIOD  
 pd.h, 275

PD\_CBL\_READY\_TIMER\_PERIOD  
 pd.h, 275

PD\_COLLISION\_SRC\_COOL\_OFF\_TIMER\_PERIOD  
 pd.h, 275

PD\_CUR\_PER\_UNIT  
 pd.h, 275

PD\_DPM\_RESP\_REC\_RESP\_PERIOD  
 pd.h, 275

PD\_EXTERNALLY\_POWERED\_BIT\_POS  
 pd.h, 276

PD\_HARD\_RESET\_TX\_TIMER\_PERIOD  
 pd.h, 276

PD\_MAX\_SRC\_CAP\_TRY  
 pd.h, 276

PD\_NO\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 276

PD\_PHY\_BUSY\_TIMER\_PERIOD  
 pd.h, 276

PD\_PPS\_SRC\_TIMER\_PERIOD  
 pd.h, 276

PD\_PS\_HARD\_RESET\_TIMER\_PERIOD  
 pd.h, 276

PD\_PS\_SNK\_TRANSITION\_TIMER\_PERIOD  
 pd.h, 276

PD\_PS\_SRC\_OFF\_TIMER\_PERIOD  
 pd.h, 276

PD\_PS\_SRC\_ON\_TIMER\_PERIOD  
 pd.h, 277

PD\_PS\_SRC\_TRANS\_TIMER\_PERIOD  
 pd.h, 277

PD\_RECEIVER\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 277

PD\_SENDER\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 277

PD\_SINK\_TX\_TIMER\_PERIOD  
 pd.h, 277

PD\_SINK\_VBUS\_TURN\_OFF\_TIMER\_PERIOD  
 pd.h, 277

PD\_SINK\_VBUS\_TURN\_ON\_TIMER\_PERIOD  
 pd.h, 277

PD\_SINK\_WAIT\_CAP\_TIMER\_PERIOD  
 pd.h, 277

PD\_SOURCE\_TRANSITION\_TIMER\_PERIOD  
 pd.h, 277

PD\_SRC\_CAP\_TIMER\_PERIOD  
 pd.h, 278

PD\_SRC\_RECOVER\_TIMER\_PERIOD  
 pd.h, 278

PD\_SWAP\_SRC\_START\_TIMER\_PERIOD  
 pd.h, 278

PD\_VBUS\_TURN\_OFF\_TIMER\_PERIOD  
 pd.h, 278

PD\_VBUS\_TURN\_ON\_TIMER\_PERIOD  
 pd.h, 278

PD\_VCONN\_OFF\_TIMER\_PERIOD  
 pd.h, 278

PD\_VCONN\_ON\_TIMER\_PERIOD  
 pd.h, 278

PD\_VCONN\_SWAP\_INITIATOR\_TIMER\_PERIOD  
 pd.h, 278

PD\_VCONN\_TURN\_ON\_TIMER\_PERIOD  
 pd.h, 278

PD\_VDM\_ENTER\_MODE\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 279

PD\_VDM\_EXIT\_MODE\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 279

PD\_VDM\_RESPONSE\_TIMER\_PERIOD  
 pd.h, 279

PD\_VOLT\_PER\_UNIT

pd.h, 279  
padval  
    dpm\_status\_t, 62  
pas\_cbl\_vdo  
    pd\_do\_t, 86  
pd.h  
    apdo\_t, 284  
    app\_evt\_t, 285  
    app\_fault\_mask\_t, 286  
    app\_req\_status\_t, 286  
    app\_resp\_cbk\_t, 283  
    app\_swap\_resp\_t, 286  
    BC\_SINK\_1\_2\_MODE\_ENABLE\_MASK, 268  
    BC\_SINK\_APPLE\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_1\_2\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_AFC\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_APPLE\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_QC\_4\_0\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_QC\_MODE\_ENABLE\_MASK, 268  
    BC\_SRC\_QC\_VER\_2\_CLASS\_A\_VAL, 268  
    BC\_SRC\_QC\_VER\_2\_CLASS\_B\_VAL, 268  
    BC\_SRC\_QC\_VER\_3\_CLASS\_A\_VAL, 269  
    BC\_SRC\_QC\_VER\_3\_CLASS\_B\_VAL, 269  
BDO\_HDR\_IDX, 269  
bist\_mode\_t, 287  
CC\_CHANNEL\_1, 269  
CC\_CHANNEL\_2, 269  
CCG\_CC\_STAT\_DRP\_TOGGLE, 269  
CCG\_CC\_STAT\_RD\_PRESENT, 269  
CCG\_CC\_STAT\_RP\_PRESENT, 269  
CCG\_CC\_STAT\_VCONN\_ACTIVE, 269  
CCG\_CC\_STAT\_ZOPEN, 270  
CCG\_DPM\_ERROR\_NO\_VCONN, 270  
CCG\_DPM\_ERROR\_NONE, 270  
CCG\_FRS\_RX\_ENABLE\_MASK, 270  
CCG\_FRS\_TX\_ENABLE\_MASK, 270  
CCG\_PD\_EXT\_PPS\_STATUS\_SIZE, 270  
CCG\_PD\_EXT\_SRCCAP\_INP\_INDEX, 270  
CCG\_PD\_EXT\_SRCCAP\_INP\_UNCONSTRAINED, 270  
CCG\_PD\_EXT\_SRCCAP\_PDP\_INDEX, 270  
CCG\_PD\_EXT\_SRCCAP\_SIZE, 271  
CCG\_PD\_EXT\_STATUS\_SIZE, 271  
CCG\_PD\_FIX\_SRC\_PDO\_MASK\_REV2, 271  
CCG\_PD\_FIX\_SRC\_PDO\_MASK\_REV3, 271  
CCG\_PD\_FLAG\_CONTRACT\_NEG\_ACTIVE, 271  
CCG\_PD\_FLAG\_EXPLICIT\_CONTRACT, 271  
CCG\_PD\_FLAG\_POWER\_SINK, 271  
CCG\_PD\_FLAG\_SRC\_READY, 271  
CY\_VID, 271  
cbl\_term\_t, 287  
cbl\_vbus\_cur\_t, 287  
ctrl\_msg\_t, 288  
DP\_SVID, 272  
DRP\_TOGGLE\_PERIOD, 272  
data\_msg\_t, 288  
dpm\_pd\_cmd\_cbk\_t, 283  
dpm\_pd\_cmd\_t, 289  
dpm\_typec\_cmd\_cbk\_t, 283  
dpm\_typec\_cmd\_resp\_t, 289  
dpm\_typec\_cmd\_t, 290  
extd\_msg\_t, 290  
fr\_swap\_supp\_t, 291  
GET\_DR\_SWAP\_RESP, 272  
GET\_PR\_SWAP\_RESP, 272  
GET\_VCONN\_SWAP\_RESP, 272  
GIVE\_BACK\_MASK, 272  
get\_pd\_config, 302  
get\_pd\_port\_config, 302  
HPD\_RX\_ACTIVITY\_TIMER\_PERIOD\_MAX, 272  
HPD\_RX\_ACTIVITY\_TIMER\_PERIOD\_MIN, 272  
I\_1P5A, 273  
I\_1A, 273  
I\_2A, 273  
I\_3A, 273  
I\_5A, 273  
ID\_HEADER\_IDX, 273  
ISAFE\_0A, 273  
ISAFE\_DEF, 273  
MAX\_CBL\_DSC\_ID\_COUNT, 273  
MAX\_EXTD\_MSG\_LEGACY\_LEN, 274  
MAX\_EXTD\_PKT\_SIZE, 274  
MAX\_EXTD\_PKT\_WORDS, 274  
MAX\_HARD\_RESET\_COUNT, 274  
MAX\_MESSAGE\_ID, 274  
MAX\_NO\_OF\_DO, 274  
MAX\_NO\_OF\_PDO, 274  
MAX\_NO\_OF\_VDO, 274  
MAX\_SOP\_TYPES, 274  
MAX\_SRC\_CAP\_COUNT, 275  
PD\_BIST\_CONT\_MODE\_TIMER\_PERIOD, 275  
PD\_CBL\_DELAY\_TIMER\_PERIOD, 275  
PD\_CBL\_DSC\_ID\_START\_TIMER\_PERIOD, 275  
PD\_CBL\_DSC\_ID\_TIMER\_PERIOD, 275  
PD\_CBL\_READY\_TIMER\_PERIOD, 275  
PD\_COLLISION\_SRC\_COOL\_OFF\_TIMER\_PERIOD, 275  
PD\_CUR\_PER\_UNIT, 275  
PD\_DPM\_RESP\_REC\_RESP\_PERIOD, 275  
PD\_EXTERNALLY\_POWERED\_BIT\_POS, 276  
PD\_HARD\_RESET\_TX\_TIMER\_PERIOD, 276  
PD\_MAX\_SRC\_CAP\_TRY, 276  
PD\_NO\_RESPONSE\_TIMER\_PERIOD, 276  
PD\_PHY\_BUSY\_TIMER\_PERIOD, 276  
PD\_PPS\_SRC\_TIMER\_PERIOD, 276  
PD\_PS\_HARD\_RESET\_TIMER\_PERIOD, 276  
PD\_PS\_SNK\_TRANSITION\_TIMER\_PERIOD, 276  
PD\_PS\_SRC\_OFF\_TIMER\_PERIOD, 276  
PD\_PS\_SRC\_ON\_TIMER\_PERIOD, 277  
PD\_PS\_SRC\_TRANS\_TIMER\_PERIOD, 277  
PD\_RECEIVER\_RESPONSE\_TIMER\_PERIOD, 277  
PD\_SENDER\_RESPONSE\_TIMER\_PERIOD, 277  
PD\_SINK\_TX\_TIMER\_PERIOD, 277

PD\_SINK\_VBUS\_TURN\_OFF\_TIMER\_PERIOD, 277  
 PD\_SINK\_VBUS\_TURN\_ON\_TIMER\_PERIOD, 277  
 PD\_SINK\_WAIT\_CAP\_TIMER\_PERIOD, 277  
 PD\_SOURCE\_TRANSITION\_TIMER\_PERIOD, 277  
 PD\_SRC\_CAP\_TIMER\_PERIOD, 278  
 PD\_SRC\_RECOVER\_TIMER\_PERIOD, 278  
 PD\_SWAP\_SRC\_START\_TIMER\_PERIOD, 278  
 PD\_VBUS\_TURN\_OFF\_TIMER\_PERIOD, 278  
 PD\_VBUS\_TURN\_ON\_TIMER\_PERIOD, 278  
 PD\_VCONN\_OFF\_TIMER\_PERIOD, 278  
 PD\_VCONN\_ON\_TIMER\_PERIOD, 278  
 PD\_VCONN\_SWAP\_INITIATOR\_TIMER\_PERIOD, 278  
 PD\_VCONN\_TURN\_ON\_TIMER\_PERIOD, 278  
 PD\_VDM\_ENTER\_MODE\_RESPONSE\_TIMER\_PERIOD, 279  
 PD\_VDM\_EXIT\_MODE\_RESPONSE\_TIMER\_PERIOD, 279  
 PD\_VDM\_RESPONSE\_TIMER\_PERIOD, 279  
 PD\_VOLT\_PER\_UNIT, 279  
 pd\_ams\_type, 291  
 pd\_cable\_reset\_reason\_t, 291  
 pd\_cbk\_t, 283  
 pd\_contract\_status\_t, 291  
 pd\_devtype\_t, 292  
 pd\_emca\_sr\_reason\_t, 292  
 pd\_err\_recov\_reason\_t, 292  
 pd\_get\_ptr\_bat\_chg\_tbl, 303  
 pd\_get\_ptr\_bb\_tbl, 303  
 pd\_get\_ptr\_chg\_cfg\_tbl, 303  
 pd\_get\_ptr\_ocp\_tbl, 304  
 pd\_get\_ptr\_otp\_tbl, 304  
 pd\_get\_ptr\_ovp\_tbl, 305  
 pd\_get\_ptr\_pwr\_tbl, 305  
 pd\_get\_ptr\_scp\_tbl, 305  
 pd\_get\_ptr\_type\_a\_chg\_cfg\_tbl, 306  
 pd\_get\_ptr\_type\_a\_pwr\_tbl, 306  
 pd\_get\_ptr\_uvp\_tbl, 306  
 pd\_get\_ptr\_vconn\_ocp\_tbl, 307  
 pd\_hard\_reset\_reason\_t, 293  
 pd\_is\_msg, 307  
 pd\_msg\_class\_t, 293  
 pd\_rev\_t, 293  
 pd\_soft\_reset\_reason\_t, 294  
 pd\_timer\_id\_t, 294  
 pdo\_t, 295  
 pe\_cbl\_state\_t, 295  
 pe\_fsm\_state\_t, 295  
 peak\_cur\_cap\_t, 297  
 port\_role\_t, 297  
 port\_type\_t, 297  
 pwr\_ready\_cbk\_t, 284  
 RDO\_IDX, 279  
 rd\_cc\_status\_t, 297  
 rdo\_type\_t, 298  
 resp\_status\_t, 298  
 rp\_cc\_status\_t, 298  
 rp\_term\_t, 299  
 SNK\_DETACH\_VBUS\_POLL\_COUNT, 279  
 SNK\_MIN\_MAX\_MASK, 279  
 SRC\_DRP\_MIN\_DC, 279  
 STD\_SVID, 279  
 STD\_VDM\_VERSION\_IDX, 280  
 STD\_VDM\_VERSION\_REV2, 280  
 STD\_VDM\_VERSION\_REV3, 280  
 STD\_VDM\_VERSION, 280  
 sink\_discharge\_off\_cbk\_t, 284  
 sop\_t, 299  
 std\_vdm\_cmd\_t, 299  
 std\_vdm\_cmd\_type\_t, 300  
 std\_vdm\_prod\_t, 300  
 std\_vdm\_ver\_t, 300  
 TBT\_SVID, 280  
 TYPEC\_CC\_DEBOUNCE\_TIMER\_PERIOD, 280  
 TYPEC\_DRP\_TRY\_TIMER\_PERIOD, 280  
 TYPEC\_ERROR\_RECOVERY\_TIMER\_PERIOD, 280  
 TYPEC\_FSM\_GENERIC, 280  
 TYPEC\_FSM\_NONE, 281  
 TYPEC\_PD3\_RPCCHANGE\_DEBOUNCE\_PERIOD, 281  
 TYPEC\_PD\_DEBOUNCE\_TIMER\_PERIOD, 281  
 TYPEC\_RD\_DEBOUNCE\_TIMER\_PERIOD, 281  
 TYPEC\_SRC\_DETACH\_DEBOUNCE\_PERIOD, 281  
 try\_src\_snk\_t, 301  
 typec\_fsm\_state\_t, 301  
 VDM\_HEADER\_IDX, 281  
 VSAFE\_0V\_HARD\_RESET, 281  
 VSAFE\_0V\_PR\_SWAP\_SNK\_SRC, 281  
 VSAFE\_0V, 281  
 VSAFE\_12V, 282  
 VSAFE\_13V, 282  
 VSAFE\_15V, 282  
 VSAFE\_19V, 282  
 VSAFE\_20V, 282  
 VSAFE\_3\_6V, 282  
 VSAFE\_5V, 282  
 VSAFE\_9V, 282  
 vdm\_ams\_t, 301  
 vdm\_resp\_cbk\_t, 284  
 vdm\_type\_t, 302  
 pd\_adc\_calibrate  
     pdss\_hal.h, 344  
 pd\_adc\_comparator\_ctrl  
     pdss\_hal.h, 344  
 pd\_adc\_comparator\_sample  
     pdss\_hal.h, 345  
 pd\_adc\_free\_run\_ctrl  
     pdss\_hal.h, 346  
 pd\_adc\_get\_comparator\_status  
     pdss\_hal.h, 346  
 pd\_adc\_get\_vbus\_voltage

pdss\_hal.h, 346  
pd\_adc\_init  
    pdss\_hal.h, 347  
pd\_adc\_level\_to\_volt  
    pdss\_hal.h, 347  
pd\_adc\_sample  
    pdss\_hal.h, 348  
pd\_adc\_volt\_to\_level  
    pdss\_hal.h, 348  
pd\_ams\_type  
    pd.h, 291  
pd\_cable\_reset\_reason\_t  
    pd.h, 291  
pd\_cbk\_t  
    pd.h, 283  
pd\_cf\_disable  
    pdss\_hal.h, 349  
pd\_cf\_enable  
    pdss\_hal.h, 349  
pd\_cf\_get\_status  
    pdss\_hal.h, 349  
pd\_cmp\_cbk\_t  
    pdss\_hal.h, 338  
pd\_cmp\_get\_status  
    pdss\_hal.h, 350  
pd\_common/dpm.h, 243  
pd\_common/pd.h, 260  
pd\_common/pd\_policy\_engine.h, 307  
pd\_common/pd\_protocol.h, 311  
pd\_common/typec\_manager.h, 318  
pd\_config\_t, 80  
    application, 81  
    flash\_checksum, 81  
    flashing\_mode, 81  
    flashing\_vid, 81  
    mfg\_info\_length, 81  
    mfg\_info\_offset, 82  
    pd\_port\_cnt, 82  
    port\_conf, 82  
    reserved\_0, 82  
    reserved\_1, 82  
    reserved\_2, 82  
    table\_checksum, 82  
    table\_sign, 82  
    table\_size, 82  
    table\_type, 83  
    table\_version, 83  
pd\_connected  
    dpm\_status\_t, 62  
pd\_contract\_info\_t, 83  
    rdo, 83  
    status, 83  
pd\_contract\_status\_t  
    pd.h, 291  
pd\_devtype\_t  
    pd.h, 292  
pd\_disabled  
    dpm\_status\_t, 63  
pd\_do\_t, 84  
    act\_cbl\_vdo, 85  
    bat\_snk, 85  
    bat\_src, 85  
    bist\_do, 85  
    dp\_cfg\_vdo, 85  
    dp\_stat\_vdo, 86  
    fixed\_snk, 86  
    fixed\_src, 86  
    pas\_cbl\_vdo, 86  
    qc\_4\_0\_data\_vdo, 86  
    rdo\_bat, 86  
    rdo\_bat\_gvb, 86  
    rdo\_fix\_var, 86  
    rdo\_fix\_var\_gvb, 86  
    rdo\_gen, 87  
    rdo\_gen\_gvb, 87  
    src\_gen, 87  
    std\_ama\_vdo, 87  
    std\_ama\_vdo\_pd3, 87  
    std\_cbl\_vdo, 87  
    std\_cert\_vdo, 87  
    std\_dp\_vdo, 87  
    std\_id\_hdr, 87  
    std\_prod\_vdo, 88  
    std\_svrid\_res, 88  
    std\_vdm\_hdr, 88  
    ustd\_qc\_4\_0\_hdr, 88  
    ustd\_vdm\_hdr, 88  
    val, 88  
    var\_snk, 88  
    var\_src, 88  
pd\_do\_t::ACT\_CBL\_VDO, 19  
    cbl\_fw\_ver, 19  
    cbl\_hw\_ver, 19  
    cbl\_latency, 19  
    cbl\_term, 19  
    max\_vbus\_volt, 20  
    rsvd1, 20  
    rsvd2, 20  
    sop\_dp, 20  
    typec\_abc, 20  
    typec\_plug, 20  
    usb\_ss\_sup, 20  
    vbus\_cur, 20  
    vbus\_thru\_cbl, 20  
    vdo\_version, 21  
pd\_do\_t::BAT\_SNK, 39  
    max\_voltage, 39  
    min\_voltage, 39  
    op\_power, 39  
    supply\_type, 39  
pd\_do\_t::BAT\_SRC, 40  
    max\_power, 40  
    max\_voltage, 40  
    min\_voltage, 40  
    supply\_type, 40  
pd\_do\_t::BIST\_DO, 45

mode, 45  
 rsvd1, 45  
 rsvd2, 45  
**pd\_do\_t::DP\_CONFIG\_VDO**, 50  
 dfp\_asgmt, 50  
 rsvd1, 50  
 rsvd2, 50  
 sel\_conf, 50  
 sign, 51  
 ufp\_asgmt, 51  
**pd\_do\_t::DP\_STATUS\_VDO**, 51  
 dfp\_upp\_conn, 51  
 en, 51  
 exit, 51  
 hpd\_irq, 52  
 hpd\_state, 52  
 mult\_fun, 52  
 pwr\_low, 52  
 rsvd, 52  
 usb\_cfg, 52  
**pd\_do\_t::FIXED\_SNK**, 68  
 dr\_swap, 69  
 dual\_role\_power, 69  
 extPowered, 69  
 fr\_swap, 69  
 high\_cap, 69  
 op\_current, 69  
 rsrvd, 69  
 supply\_type, 70  
 usb\_comm\_cap, 70  
 voltage, 70  
**pd\_do\_t::FIXED\_SRC**, 70  
 dr\_swap, 70  
 dual\_role\_power, 70  
 extPowered, 71  
 max\_current, 71  
 pk\_current, 71  
 reserved, 71  
 supply\_type, 71  
 unchunk\_sup, 71  
 usb\_comm\_cap, 71  
 usb\_suspend\_sup, 71  
 voltage, 71  
**pd\_do\_t::PAS\_CBL\_VDO**, 79  
 cbl\_fw\_ver, 79  
 cbl\_hw\_ver, 79  
 cbl\_latency, 79  
 cbl\_term, 79  
 max\_vbus\_volt, 79  
 rsvd1, 79  
 rsvd2, 80  
 rsvd3, 80  
 typec\_abc, 80  
 typec\_plug, 80  
 usb\_ss\_sup, 80  
 vbus\_cur, 80  
 vdo\_version, 80  
**pd\_do\_t::QC\_4\_0\_DATA\_VDO**, 109  
 data\_0, 109  
 data\_1, 109  
 data\_2, 109  
 data\_3, 110  
**pd\_do\_t::RDO\_BAT\_GIVEBACK**, 111  
 cap\_mismatch, 112  
 give\_back\_flag, 112  
 min\_op\_power, 112  
 no\_usb\_suspend, 112  
 obj\_pos, 112  
 op\_power, 112  
 rsrvd1, 112  
 rsrvd2, 112  
 unchunk\_sup, 113  
 usb\_comm\_cap, 113  
**pd\_do\_t::RDO\_BAT**, 110  
 cap\_mismatch, 110  
 give\_back\_flag, 110  
 max\_op\_power, 110  
 no\_usb\_suspend, 110  
 obj\_pos, 111  
 op\_power, 111  
 rsrvd1, 111  
 rsrvd2, 111  
 unchunk\_sup, 111  
 usb\_comm\_cap, 111  
**pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK**, 114  
 cap\_mismatch, 115  
 give\_back\_flag, 115  
 min\_op\_current, 115  
 no\_usb\_suspend, 115  
 obj\_pos, 115  
 op\_current, 115  
 rsrvd1, 115  
 rsrvd2, 116  
 unchunk\_sup, 116  
 usb\_comm\_cap, 116  
**pd\_do\_t::RDO\_FIXED\_VAR**, 113  
 cap\_mismatch, 113  
 give\_back\_flag, 113  
 max\_op\_current, 113  
 no\_usb\_suspend, 114  
 obj\_pos, 114  
 op\_current, 114  
 rsrvd1, 114  
 rsrvd2, 114  
 unchunk\_sup, 114  
 usb\_comm\_cap, 114  
**pd\_do\_t::RDO\_GEN\_GVB**, 118  
 cap\_mismatch, 118  
 give\_back\_flag, 118  
 max\_power\_cur, 118  
 no\_usb\_suspend, 118  
 obj\_pos, 118  
 op\_power\_cur, 118  
 rsrvd1, 118  
 rsrvd2, 119  
 unchunk\_sup, 119

usb\_comm\_cap, 119  
pd\_do\_t::RDO\_GEN, 116  
cap\_mismatch, 116  
give\_back\_flag, 116  
min\_max\_power\_cur, 117  
no\_usb\_suspend, 117  
obj\_pos, 117  
op\_power\_cur, 117  
rsrvd1, 117  
rsrvd2, 117  
unchunk\_sup, 117  
usb\_comm\_cap, 117  
pd\_do\_t::SRC\_GEN, 121  
max\_cur\_power, 121  
max\_voltage, 121  
min\_voltage, 121  
supply\_type, 121  
pd\_do\_t::STD\_AMA\_VDO\_PD3, 123  
ama\_fw\_ver, 123  
ama\_hw\_ver, 123  
rsrvd1, 124  
usb\_ss\_sup, 124  
vbus\_req, 124  
vcon\_pwr, 124  
vcon\_req, 124  
vdo\_version, 124  
pd\_do\_t::STD\_AMA\_VDO, 122  
ama\_fw\_ver, 122  
ama\_hw\_ver, 122  
rsrvd1, 122  
ssrx1, 122  
ssrx2, 122  
sstx1, 122  
sstx2, 122  
usb\_ss\_sup, 123  
vbus\_req, 123  
vcon\_pwr, 123  
vcon\_req, 123  
pd\_do\_t::STD\_CBL\_VDO, 124  
cbl\_fw\_ver, 125  
cbl\_hw\_ver, 125  
cbl\_latency, 125  
cbl\_term, 125  
rsrvd1, 125  
sop\_dp, 125  
ssrx1, 125  
ssrx2, 125  
sstx1, 126  
sstx2, 126  
typec\_abc, 126  
typec\_plug, 126  
usb\_ss\_sup, 126  
vbus\_cur, 126  
vbus\_thru\_cbl, 126  
pd\_do\_t::STD\_CERT\_VDO, 126  
usb\_xid, 127  
pd\_do\_t::STD\_DP\_VDO, 127  
dfp\_d\_pin, 127  
port\_cap, 127  
recep, 127  
rsvd, 127  
signal, 128  
ufp\_d\_pin, 128  
usb2\_0, 128  
pd\_do\_t::STD\_PROD\_VDO, 128  
bcd\_dev, 128  
usb\_pid, 128  
pd\_do\_t::STD\_SVID\_RESP\_VDO, 129  
svid\_n, 129  
svid\_n1, 129  
pd\_do\_t::STD\_VDM\_HDR, 129  
cmd, 129  
cmd\_type, 129  
obj\_pos, 130  
rsrvd1, 130  
rsrvd2, 130  
st\_ver, 130  
svid, 130  
vdm\_type, 130  
pd\_do\_t::STD\_VDM\_ID\_HDR, 130  
mod\_support, 131  
prod\_type, 131  
usb\_dev, 131  
usb\_host, 131  
usb\_vid, 131  
pd\_do\_t::USTD\_QC\_4\_0\_HDR, 148  
cmd\_0, 148  
cmd\_1, 148  
svid, 148  
vdm\_type, 148  
pd\_do\_t::USTD\_VDM\_HDR, 148  
cmd, 149  
cmd\_type, 149  
rsrvd1, 149  
seq\_num, 149  
svid, 149  
vdm\_type, 149  
vdm\_ver, 149  
pd\_do\_t::VAR\_SNK, 150  
max\_voltage, 151  
min\_voltage, 151  
op\_current, 151  
supply\_type, 151  
pd\_do\_t::VAR\_SRC, 151  
max\_current, 151  
max\_voltage, 151  
min\_voltage, 152  
supply\_type, 152  
pd\_emca\_sr\_reason\_t  
pd.h, 292  
pd\_err\_recov\_reason\_t  
pd.h, 292  
pd\_extd\_hdr\_t, 89  
chunk\_no, 89  
chunked, 89  
data\_size, 89

extd, 89  
 request, 89  
 rsvd1, 89  
 val, 90  
**pd\_get\_ptr\_bat\_chg\_tbl**  
 pd.h, 303  
**pd\_get\_ptr\_bb\_tbl**  
 pd.h, 303  
**pd\_get\_ptr\_chg\_cfg\_tbl**  
 pd.h, 303  
**pd\_get\_ptr\_ocp\_tbl**  
 pd.h, 304  
**pd\_get\_ptr\_otp\_tbl**  
 pd.h, 304  
**pd\_get\_ptr\_ovp\_tbl**  
 pd.h, 305  
**pd\_get\_ptr\_pwr\_tbl**  
 pd.h, 305  
**pd\_get\_ptr\_scp\_tbl**  
 pd.h, 305  
**pd\_get\_ptr\_type\_a\_chg\_cfg\_tbl**  
 pd.h, 306  
**pd\_get\_ptr\_type\_a\_pwr\_tbl**  
 pd.h, 306  
**pd\_get\_ptr\_upv\_tbl**  
 pd.h, 306  
**pd\_get\_ptr\_vconn\_ocp\_tbl**  
 pd.h, 307  
**pd\_get\_vbus\_adc\_level**  
 pdss\_hal.h, 350  
**pd\_hal/hal\_ccgx.h**, 322  
**pd\_hal/hpd.h**, 326  
**pd\_hal/pdss\_hal.h**, 330  
**pd\_hal\_abort\_auto\_toggle**  
 pdss\_hal.h, 351  
**pd\_hal\_cleanup**  
 pdss\_hal.h, 351  
**pd\_hal\_config\_auto\_toggle**  
 pdss\_hal.h, 351  
**pd\_hal\_dual\_fet\_config**  
 pdss\_hal.h, 352  
**pd\_hal\_get\_vbus\_detach\_adc**  
 pdss\_hal.h, 352  
**pd\_hal\_get\_vbus\_detach\_input**  
 pdss\_hal.h, 352  
**pd\_hal\_init**  
 pdss\_hal.h, 353  
**pd\_hal\_is\_auto\_toggle\_active**  
 pdss\_hal.h, 353  
**pd\_hal\_measure\_vbus**  
 pdss\_hal.h, 353  
**pd\_hal\_set\_fet\_drive**  
 pdss\_hal.h, 354  
**pd\_hal\_set\_supply\_change\_evt\_cb**  
 pdss\_hal.h, 354  
**pd\_hal\_set\_vbus\_detach\_params**  
 pdss\_hal.h, 354  
**pd\_hal\_set\_vbus\_mon\_divider**

pdss\_hal.h, 355  
**pd\_hal\_typeec\_sm\_restart**  
 pdss\_hal.h, 355  
**pd\_hal\_vconn\_ocp\_disable**  
 pdss\_hal.h, 355  
**pd\_hal\_vconn\_ocp\_enable**  
 pdss\_hal.h, 356  
**pd\_hard\_reset\_reason\_t**  
 pd.h, 293  
**pd\_hdr\_t**, 92  
 hdr, 92  
 val, 92  
**pd\_hdr\_t::PD\_HDR**, 90  
 chunk\_no, 90  
 chunked, 90  
 data\_role, 90  
 data\_size, 90  
 extd, 91  
 len, 91  
 msg\_id, 91  
 msg\_type, 91  
 pwr\_role, 91  
 request, 91  
 rsvd1, 91  
 spec\_rev, 91  
**pd\_is\_msg**  
 pd.h, 307  
**pd\_is\_vconn\_present**  
 pdss\_hal.h, 356  
**pd\_lscsa\_calc\_cfg**  
 pdss\_hal.h, 356  
**pd\_lscsa\_cfg**  
 pdss\_hal.h, 357  
**pd\_msg\_class\_t**  
 pd.h, 293  
**pd\_operation\_en**  
 pd\_port\_config\_t, 99  
**pd\_packet\_extd\_t**, 92  
 dat, 93  
 data\_role, 93  
 hdr, 93  
 len, 93  
 msg, 93  
 sop, 93  
**pd\_packet\_t**, 93  
 dat, 94  
 data\_role, 94  
 hdr, 94  
 len, 94  
 msg, 94  
 sop, 94  
**pd\_phy\_abort\_bist\_cm2**  
 pdss\_hal.h, 357  
**pd\_phy\_abort\_tx\_msg**  
 pdss\_hal.h, 357  
**pd\_phy\_cbk\_t**  
 pdss\_hal.h, 338  
**pd\_phy\_deepsleep**

pdss\_hal.h, 358  
pd\_phy\_dis\_unchunked\_tx  
    pdss\_hal.h, 358  
pd\_phy\_en\_unchunked\_tx  
    pdss\_hal.h, 358  
pd\_phy\_evt\_t  
    pdss\_hal.h, 343  
pd\_phy\_get\_rx\_packet  
    pdss\_hal.h, 359  
pd\_phy\_init  
    pdss\_hal.h, 359  
pd\_phy\_is\_busy  
    pdss\_hal.h, 359  
pd\_phy\_load\_msg  
    pdss\_hal.h, 361  
pd\_phy\_refresh\_roles  
    pdss\_hal.h, 361  
pd\_phy\_reset\_rx\_tx\_sm  
    pdss\_hal.h, 362  
pd\_phy\_send\_bist\_cm2  
    pdss\_hal.h, 362  
pd\_phy\_send\_msg  
    pdss\_hal.h, 362  
pd\_phy\_send\_reset  
    pdss\_hal.h, 362  
pd\_phy\_wakeup  
    pdss\_hal.h, 363  
pd\_policy\_engine.h  
    get\_pe\_state\_buf, 308  
    get\_spec\_rev\_determined, 308  
    pe\_clear\_hard\_reset\_count, 308  
    pe\_disabled, 309  
    pe\_fsm, 309  
    pe\_get\_pps\_status, 309  
    pe\_init, 310  
    pe\_is\_busy, 310  
    pe\_push\_to\_buf, 310  
    pe\_start, 310  
    pe\_stop, 311  
pd\_port\_cnt  
    pd\_config\_t, 82  
pd\_port\_config\_t, 95  
    bat\_chg\_tbl\_offset, 96  
    bb\_tbl\_offset, 96  
    cable\_disc\_en, 96  
    chg\_cfg\_tbl\_offset, 96  
    current\_level, 96  
    dead\_bat\_support, 96  
    default\_role, 97  
    default\_sink\_pdo\_mask, 97  
    default\_src\_pdo\_mask, 97  
    dock\_cfg\_tbl\_offset, 97  
    dp\_config\_supported, 97  
    dp\_mode\_trigger, 97  
    dp\_mux\_control, 97  
    dp\_oper, 97  
    dp\_pref\_mode, 97  
    drp\_toggle\_en, 98  
err\_recovery\_en, 98  
ext\_src\_cap\_length, 98  
ext\_src\_cap\_offset, 98  
frs\_enable, 98  
id\_vdm\_length, 98  
id\_vdm\_offset, 98  
is\_snk\_bat, 98  
is\_src\_bat, 98  
mode\_vdm\_length, 99  
mode\_vdm\_offset, 99  
ocp\_tbl\_offset, 99  
otp\_tbl\_offset, 99  
ovp\_tbl\_offset, 99  
pd\_operation\_en, 99  
port\_disable, 99  
port\_role, 99  
protection\_enable, 99  
pwr\_tbl\_offset, 100  
ra\_timeout, 100  
reserved\_0, 100  
reserved\_1, 100  
reserved\_10, 100  
reserved\_11, 100  
reserved\_3, 100  
reserved\_4, 100  
reserved\_6, 100  
reserved\_8, 101  
reserved\_9, 101  
rp\_supported, 101  
scp\_tbl\_offset, 101  
snk\_pdo\_cnt, 101  
snk\_pdo\_list, 101  
snk\_pdo\_max\_min\_current\_pwr, 101  
snk\_usb\_comm\_en, 101  
snk\_usb\_susp\_en, 101  
spm\_cfg\_tbl\_offset, 102  
src\_pdo\_cnt, 102  
src\_pdo\_list, 102  
svid\_vdm\_length, 102  
svid\_vdm\_offset, 102  
swap\_response, 102  
try\_src\_en, 102  
type\_a\_chg\_tbl\_offset, 102  
type\_a\_enable, 102  
type\_a\_pwr\_tbl\_offset, 103  
uvp\_tbl\_offset, 103  
vconn\_ocp\_tbl\_offset, 103  
vconn\_retain, 103  
pd\_port\_status\_t, 106  
    status, 106  
    val, 106  
pd\_port\_status\_t::PD\_PORT\_STAT, 103  
    ccg\_spec\_rev, 104  
    contract\_exist, 104  
    cur\_data\_role, 104  
    cur\_power\_role, 104  
    dfilt\_data\_pref, 104  
    dfilt\_data\_role, 104

dfilt\_power\_pref, 104  
 dfilt\_power\_role, 104  
 emca\_present, 104  
 emca\_spec\_rev, 105  
 min\_state, 105  
 pe\_rdy, 105  
 peer\_pd3\_supp, 105  
 peer\_unchunk\_supp, 105  
 reserved0, 105  
 reserved2, 105  
 rp\_status, 105  
 vconn\_on, 105  
 vconn\_src, 106  
**pd\_power\_status\_t**, 107  
 battery\_input, 107  
 dummy, 107  
 event\_flags, 107  
 intl\_temperature, 107  
 present\_input, 107  
 temp\_status, 107  
**pd\_prot\_dis\_bist\_cm2**  
 pd\_protocol.h, 312  
**pd\_prot\_dis\_bist\_test\_data**  
 pd\_protocol.h, 312  
**pd\_prot\_en\_bist\_cm2**  
 pd\_protocol.h, 312  
**pd\_prot\_en\_bist\_test\_data**  
 pd\_protocol.h, 313  
**pd\_prot\_get\_rx\_packet**  
 pd\_protocol.h, 313  
**pd\_prot\_init**  
 pd\_protocol.h, 313  
**pd\_prot\_is\_busy**  
 pd\_protocol.h, 314  
**pd\_prot\_refresh\_roles**  
 pd\_protocol.h, 314  
**pd\_prot\_reset**  
 pd\_protocol.h, 314  
**pd\_prot\_reset\_all**  
 pd\_protocol.h, 315  
**pd\_prot\_reset\_rx**  
 pd\_protocol.h, 315  
**pd\_prot\_rx\_dis**  
 pd\_protocol.h, 315  
**pd\_prot\_rx\_en**  
 pd\_protocol.h, 316  
**pd\_prot\_send\_cable\_reset**  
 pd\_protocol.h, 316  
**pd\_prot\_send\_ctrl\_msg**  
 pd\_protocol.h, 316  
**pd\_prot\_send\_data\_msg**  
 pd\_protocol.h, 317  
**pd\_prot\_send\_hard\_reset**  
 pd\_protocol.h, 317  
**pd\_prot\_set\_avoid\_retry**  
 pd\_protocol.h, 317  
**pd\_prot\_start**  
 pd\_protocol.h, 318  
**pd\_prot\_stop**  
 pd\_protocol.h, 318  
**pd\_prot\_dis\_bist\_cm2**, 312  
**pd\_prot\_dis\_bist\_test\_data**, 312  
**pd\_prot\_en\_bist\_cm2**, 312  
**pd\_prot\_en\_bist\_test\_data**, 313  
**pd\_prot\_get\_rx\_packet**, 313  
**pd\_prot\_init**, 313  
**pd\_prot\_is\_busy**, 314  
**pd\_prot\_refresh\_roles**, 314  
**pd\_prot\_reset**, 314  
**pd\_prot\_reset\_all**, 315  
**pd\_prot\_reset\_rx**, 315  
**pd\_prot\_rx\_dis**, 315  
**pd\_prot\_rx\_en**, 316  
**pd\_prot\_send\_cable\_reset**, 316  
**pd\_prot\_send\_ctrl\_msg**, 316  
**pd\_prot\_send\_data\_msg**, 317  
**pd\_prot\_send\_hard\_reset**, 317  
**pd\_prot\_set\_avoid\_retry**, 317  
**pd\_prot\_start**, 318  
**pd\_prot\_stop**, 318  
**pd\_rev\_t**  
 pd.h, 293  
**pd\_set\_pfc\_comp**  
 pdss\_hal.h, 363  
**pd\_set\_sr\_comp**  
 pdss\_hal.h, 364  
**pd\_soft\_reset\_reason\_t**  
 pd.h, 294  
**pd\_stop\_pfc\_comp**  
 pdss\_hal.h, 364  
**pd\_stop\_sr\_comp**  
 pdss\_hal.h, 364  
**pd\_supply\_change\_cbk\_t**  
 pdss\_hal.h, 338  
**pd\_support**  
 dpm\_status\_t, 63  
**pd\_timer\_id\_t**  
 pd.h, 294  
**pd\_typec\_dis\_dpslp\_rp**  
 pdss\_hal.h, 365  
**pd\_typec\_dis\_rd**  
 pdss\_hal.h, 365  
**pd\_typec\_dis\_rp**  
 pdss\_hal.h, 365  
**pd\_typec\_en\_dpslp\_rp**  
 pdss\_hal.h, 366  
**pd\_typec\_en\_rd**  
 pdss\_hal.h, 366  
**pd\_typec\_en\_rp**  
 pdss\_hal.h, 366  
**pd\_typec\_get\_cc\_status**  
 pdss\_hal.h, 367  
**pd\_typec\_init**  
 pdss\_hal.h, 367  
**pd\_typec\_rd\_enable**

pdss\_hal.h, 367  
pd\_typec\_set\_polarity  
    pdss\_hal.h, 367  
pd\_typec\_snk\_update\_trim  
    pdss\_hal.h, 368  
pd\_typec\_start  
    pdss\_hal.h, 368  
pd\_typec\_stop  
    pdss\_hal.h, 368  
pd\_vconn\_disable  
    pdss\_hal.h, 369  
pd\_vconn\_enable  
    pdss\_hal.h, 369  
pdo.h  
    eval\_rdo, 204  
    eval\_src\_cap, 205  
pdo\_t  
    pd.h, 295  
pdss\_hal.h  
    aux\_resistor\_config\_t, 339  
    ccg\_supply\_t, 339  
    comp\_id\_t, 340  
    comp\_tr\_id\_t, 340  
    dpdm\_mux\_cfg\_t, 340  
    FRS\_TX\_SWAP\_CTRL1\_DFLT\_VAL, 335  
    filter\_edge\_detect\_cfg\_t, 341  
    filter\_id\_t, 341  
    HEADER\_INFO\_CFG, 335  
    lscsa\_app\_config\_t, 341  
    PD\_ADC\_CB\_T, 338  
    PD\_ADC\_ID\_T, 342  
    PD\_ADC\_INPUT\_T, 342  
    PD\_ADC\_INT\_T, 342  
    pd\_adc\_calibrate, 344  
    pd\_adc\_comparator\_ctrl, 344  
    pd\_adc\_comparator\_sample, 345  
    pd\_adc\_free\_run\_ctrl, 346  
    pd\_adc\_get\_comparator\_status, 346  
    pd\_adc\_get\_vbus\_voltage, 346  
    pd\_adc\_init, 347  
    pd\_adc\_level\_to\_volt, 347  
    pd\_adc\_sample, 348  
    pd\_adc\_volt\_to\_level, 348  
    pd\_cf\_disable, 349  
    pd\_cf\_enable, 349  
    pd\_cf\_get\_status, 349  
    pd\_cmp\_cbk\_t, 338  
    pd\_cmp\_get\_status, 350  
    pd\_get\_vbus\_adc\_level, 350  
    pd\_hal\_abort\_auto\_toggle, 351  
    pd\_hal\_cleanup, 351  
    pd\_hal\_config\_auto\_toggle, 351  
    pd\_hal\_dual\_fet\_config, 352  
    pd\_hal\_get\_vbus\_detach\_adc, 352  
    pd\_hal\_get\_vbus\_detach\_input, 352  
    pd\_hal\_init, 353  
    pd\_hal\_is\_auto\_toggle\_active, 353  
    pd\_hal\_measure\_vbus, 353  
pd\_hal\_set\_fet\_drive, 354  
pd\_hal\_set\_supply\_change\_evt\_cb, 354  
pd\_hal\_set\_vbus\_detach\_params, 354  
pd\_hal\_set\_vbus\_mon\_divider, 355  
pd\_hal\_typec\_sm\_restart, 355  
pd\_hal\_vconn\_ocp\_disable, 355  
pd\_hal\_vconn\_ocp\_enable, 356  
pd\_is\_vconn\_present, 356  
pd\_lscsa\_calc\_cfg, 356  
pd\_lscsa\_cfg, 357  
pd\_phy\_abort\_bist\_cm2, 357  
pd\_phy\_abort\_tx\_msg, 357  
pd\_phy\_cbk\_t, 338  
pd\_phy\_deepsleep, 358  
pd\_phy\_dis\_unchunked\_tx, 358  
pd\_phy\_en\_unchunked\_tx, 358  
pd\_phy\_evt\_t, 343  
pd\_phy\_get\_rx\_packet, 359  
pd\_phy\_init, 359  
pd\_phy\_is\_busy, 359  
pd\_phy\_load\_msg, 361  
pd\_phy\_refresh\_roles, 361  
pd\_phy\_reset\_rx\_tx\_sm, 362  
pd\_phy\_send\_bist\_cm2, 362  
pd\_phy\_send\_msg, 362  
pd\_phy\_send\_reset, 362  
pd\_phy\_wakeup, 363  
pd\_set\_pfc\_comp, 363  
pd\_set\_sr\_comp, 364  
pd\_stop\_pfc\_comp, 364  
pd\_stop\_sr\_comp, 364  
pd\_supply\_change\_cbk\_t, 338  
pd\_typec\_dis\_dpslp\_rp, 365  
pd\_typec\_dis\_rd, 365  
pd\_typec\_dis\_rp, 365  
pd\_typec\_en\_dpslp\_rp, 366  
pd\_typec\_en\_rd, 366  
pd\_typec\_en\_rp, 366  
pd\_typec\_get\_cc\_status, 367  
pd\_typec\_init, 367  
pd\_typec\_rd\_enable, 367  
pd\_typec\_set\_polarity, 367  
pd\_typec\_snk\_update\_trim, 368  
pd\_typec\_start, 368  
pd\_typec\_stop, 368  
pd\_vconn\_disable, 369  
pd\_vconn\_enable, 369  
RCV\_INTR\_MASK, 335  
RST\_TX\_INTERRUPTS, 336  
RX\_CC\_CFG, 336  
RX\_INTERRUPTS, 336  
RX\_ORDER\_SET\_CTRL\_CFG, 336  
sbu\_switch\_state\_t, 343  
soln\_no\_vsys\_handler, 369  
soln\_vsys\_removal\_handler, 370  
TX\_INTERRUPTS, 336  
VBUS\_OCP\_GPIO\_ACTIVE\_HIGH, 337  
VBUS\_OCP\_GPIO\_ACTIVE\_LOW, 337

**VBUS\_OCP\_MODE\_EXT**, 337  
**VBUS\_OCP\_MODE\_INT\_AUTOCTRL**, 337  
**VBUS\_OCP\_MODE\_INT\_SW\_DB**, 337  
**VBUS\_OCP\_MODE\_INT**, 337  
**VBUS\_OCP\_MODE\_POLLING**, 337  
**vbus\_cf\_cbk\_t**, 339  
**vbus\_ovp\_mode\_t**, 343  
**vbus\_uvp\_mode\_t**, 344  
**pe\_cbl\_state\_t**  
 pd.h, 295  
**pe\_clear\_hard\_reset\_count**  
 pd\_policy\_engine.h, 308  
**pe\_disabled**  
 pd\_policy\_engine.h, 309  
**pe\_evt**  
 dpm\_status\_t, 63  
**pe\_fsm**  
 pd\_policy\_engine.h, 309  
**pe\_fsm\_state**  
 dpm\_status\_t, 63  
**pe\_fsm\_state\_t**  
 pd.h, 295  
**pe\_get\_pps\_status**  
 pd\_policy\_engine.h, 309  
**pe\_init**  
 pd\_policy\_engine.h, 310  
**pe\_is\_busy**  
 pd\_policy\_engine.h, 310  
**pe\_push\_to\_buf**  
 pd\_policy\_engine.h, 310  
**pe\_rdy**  
 pd\_port\_status\_t::PD\_PORT\_STAT, 105  
**pe\_start**  
 pd\_policy\_engine.h, 310  
**pe\_stop**  
 pd\_policy\_engine.h, 311  
**peak\_cur\_cap\_t**  
 pd.h, 297  
**peer\_pd3\_supp**  
 pd\_port\_status\_t::PD\_PORT\_STAT, 105  
**peer\_unchunk\_supp**  
 pd\_port\_status\_t::PD\_PORT\_STAT, 105  
**period**  
 ccg\_timer\_t, 47  
**pk\_current**  
 pd\_do\_t::FIXED\_SRC, 71  
**polarity**  
 dpm\_status\_t, 63  
**port\_cap**  
 pd\_do\_t::STD\_DP\_VDO, 127  
**port\_conf**  
 pd\_config\_t, 82  
**port\_disable**  
 dpm\_status\_t, 63  
 pd\_port\_config\_t, 99  
**port\_role**  
 dpm\_status\_t, 63  
 pd\_port\_config\_t, 99  
**port\_role\_t**  
 pd.h, 297  
**port\_status**  
 dpm\_status\_t, 63  
**port\_type\_t**  
 pd.h, 297  
**pps\_status**  
 dpm\_status\_t, 63  
**preamble**  
 usb\_i2cm\_t, 139  
**preamble\_state**  
 usb\_i2cm\_t, 139  
**present\_input**  
 pd\_power\_status\_t, 107  
**prev\_state**  
 usb\_handle\_t, 137  
**pro\_dock\_detect**  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 146  
**prod\_type**  
 pd\_do\_t::STD\_VDM\_ID\_HDR, 131  
**protection\_enable**  
 pd\_port\_config\_t, 99  
**psink.h**  
 psnk\_disable, 205  
 psnk\_enable, 206  
 psnk\_set\_current, 206  
 psnk\_set\_voltage, 206  
**psnk\_cur**  
 app\_status\_t, 35  
**psnk\_disable**  
 app\_cbk\_t, 30  
 psink.h, 205  
**psnk\_enable**  
 app\_cbk\_t, 30  
 psink.h, 206  
**psnk\_set\_current**  
 app\_cbk\_t, 31  
 psink.h, 206  
**psnk\_set\_voltage**  
 app\_cbk\_t, 31  
 psink.h, 206  
**psnk\_volt**  
 app\_status\_t, 35  
**psource.h**  
 psrc\_disable, 207  
 psrc\_enable, 208  
 psrc\_get\_voltage, 208  
 psrc\_set\_current, 208  
 psrc\_set\_voltage, 209  
**psrc\_disable**  
 app\_cbk\_t, 31  
 psource.h, 207  
**psrc\_enable**  
 app\_cbk\_t, 31  
 psource.h, 208  
**psrc\_get\_voltage**  
 psource.h, 208  
**psrc\_rising**

app\_status\_t, 35  
psrc\_set\_current  
    app\_cbk\_t, 31  
    psource.h, 208  
psrc\_set\_voltage  
    app\_cbk\_t, 31  
    psource.h, 209  
psrc\_volt  
    app\_status\_t, 35  
psrc\_volt\_old  
    app\_status\_t, 35  
pwr  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 146  
pwr\_low  
    pd\_do\_t::DP\_STATUS\_VDO, 52  
pwr\_params\_t, 108  
    fb\_ctrl\_r1, 108  
    fb\_ctrl\_r2, 108  
    fb\_type, 108  
    table\_len, 108  
    vbus\_dflt\_VOLT, 109  
    vbus\_max\_VOLT, 109  
    vbus\_min\_VOLT, 109  
pwr\_ready\_cbk  
    app\_status\_t, 36  
pwr\_ready\_cbk\_t  
    pd.h, 284  
pwr\_role  
    pd\_hdr\_t::PD\_HDR, 91  
pwr\_tbl\_offset  
    pd\_port\_config\_t, 100  
  
qc\_4\_0\_data\_vdo  
    pd\_do\_t, 86  
qc\_src\_type  
    chg\_cfg\_params\_t, 48  
queue\_disable  
    bb\_handle\_t, 41  
queue\_enable  
    bb\_handle\_t, 41  
queue\_i2cm\_enable  
    bb\_handle\_t, 42  
  
RCV\_INTR\_MASK  
    pdss\_hal.h, 335  
RDO\_IDX  
    pd.h, 279  
REV\_BYTE\_ORDER  
    utils.h, 407  
RIDGE\_SLAVE\_ADDR\_MASK  
    ridge\_slave.h, 210  
RST\_TX\_INTERRUPTS  
    pdss\_hal.h, 336  
RX\_CC\_CFG  
    pdss\_hal.h, 336  
RX\_INTERRUPTS  
    pdss\_hal.h, 336  
RX\_ORDER\_SET\_CTRL\_CFG  
    pdss\_hal.h, 336  
  
ra\_present  
    dpm\_status\_t, 64  
ra\_timeout  
    pd\_port\_config\_t, 100  
rd\_cc\_status\_t  
    pd.h, 297  
rdo  
    pd\_contract\_info\_t, 83  
rdo\_bat  
    pd\_do\_t, 86  
rdo\_bat\_gvb  
    pd\_do\_t, 86  
rdo\_fix\_var  
    pd\_do\_t, 86  
rdo\_fix\_var\_gvb  
    pd\_do\_t, 86  
rdo\_gen  
    pd\_do\_t, 87  
rdo\_gen\_gvb  
    pd\_do\_t, 87  
rdo\_type\_t  
    pd.h, 298  
recep  
    pd\_do\_t::STD\_DP\_VDO, 127  
reg\_alt\_mode\_mngr  
    alt\_modes\_mngr.h, 169  
reg\_dp\_modes  
    dp\_sid.h, 172  
reg\_intel\_modes  
    intel\_vid.h, 177  
req\_status  
    app\_resp\_t, 32  
request  
    pd\_extd\_hdr\_t, 89  
    pd\_hdr\_t::PD\_HDR, 91  
reserved  
    fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
    pd\_do\_t::FIXED\_SRC, 71  
reserved0  
    pd\_port\_status\_t::PD\_PORT\_STAT, 105  
reserved1  
    fw\_img\_status\_t::fw\_mode\_reason\_t, 73  
    sys\_fw\_metadata\_t, 133  
reserved2  
    pd\_port\_status\_t::PD\_PORT\_STAT, 105  
    sys\_fw\_metadata\_t, 133  
reserved\_0  
    bat\_chg\_params\_t, 38  
    chg\_cfg\_params\_t, 48  
    otp\_settings\_t, 77  
    pd\_config\_t, 82  
    pd\_port\_config\_t, 100  
    vconn\_ocp\_settings\_t, 152  
reserved\_1  
    bat\_chg\_params\_t, 38  
    bb\_settings\_t, 44  
    chg\_cfg\_params\_t, 48  
    pd\_config\_t, 82

pd\_port\_config\_t, 100  
 reserved\_10  
     pd\_port\_config\_t, 100  
 reserved\_11  
     pd\_port\_config\_t, 100  
 reserved\_2  
     pd\_config\_t, 82  
 reserved\_3  
     dpm\_status\_t, 64  
     pd\_port\_config\_t, 100  
 reserved\_4  
     pd\_port\_config\_t, 100  
 reserved\_6  
     pd\_port\_config\_t, 100  
 reserved\_8  
     pd\_port\_config\_t, 101  
 reserved\_9  
     pd\_port\_config\_t, 101  
 reset\_alt\_mode\_info  
     alt\_modes\_mngr.h, 169  
 resp\_buf  
     vdm\_resp\_t, 154  
 resp\_do  
     app\_resp\_t, 33  
 resp\_status\_t  
     pd.h, 298  
 restart\_volt  
     otp\_settings\_t, 77  
 retry\_cnt  
     ocp\_settings\_t, 76  
     ovp\_settings\_t, 78  
     scp\_settings\_t, 120  
     uvp\_settings\_t, 150  
 rev\_pol  
     dpm\_status\_t, 64  
 ridge\_eval\_cmd  
     intel\_ridge.h, 174  
 ridge\_reg\_reset  
     ridge\_slave.h, 210  
 ridge\_reg\_t, 119  
     ridge\_stat, 119  
     usbpd\_cmd\_reg, 120  
     val, 120  
 ridge\_reg\_t::USBPD\_CMD\_REG, 141  
     dp\_host\_conn, 142  
     hpd\_irq, 142  
     hpd\_lvl, 142  
     i2c\_int\_ack, 142  
     irq\_ack, 142  
     rsvd2, 142  
     rsvd3, 143  
     rsvd4, 143  
     soft\_rst, 143  
     tbt\_host\_conn, 143  
     usb\_host\_conn, 143  
 ridge\_reg\_t::USBPD\_STATUS\_REG, 143  
     act\_link\_train, 144  
     active\_cbl, 144  
     cbl\_type, 144  
     conn\_orien, 144  
     data\_conn\_pres, 144  
     dbg\_acc\_mode, 145  
     dbg\_alt\_m\_conn, 145  
     dp\_conn, 145  
     dp\_pin\_assign, 145  
     dp\_role, 145  
     force\_lsx, 145  
     hpd\_irq, 145  
     hpd\_lvl, 145  
     interrupt\_ack, 145  
     irq\_ack, 146  
     ovc\_indn, 146  
     pro\_dock\_detect, 146  
     pwr, 146  
     rsvd, 146  
     rsvd4, 146  
     tbt\_cbl\_gen, 146  
     tbt\_cbl\_spd, 146  
     tbt\_conn, 146  
     tbt\_type, 147  
     usb2\_conn, 147  
     usb3\_conn, 147  
     usb3\_speed, 147  
     usb\_dr, 147  
 ridge\_set\_ctrl\_change\_cb  
     intel\_ridge.h, 174  
 ridge\_set\_mux  
     intel\_ridge.h, 174  
 ridge\_slave.h  
     RIDGE\_SLAVE\_ADDR\_MASK, 210  
     ridge\_reg\_reset, 210  
     ridge\_slave\_init, 211  
     ridge\_slave\_is\_host\_connected, 211  
     ridge\_slave\_reg\_addr\_t, 210  
     ridge\_slave\_sleep, 212  
     ridge\_slave\_status\_update, 212  
     ridge\_slave\_task, 212  
     ridge\_slave\_update\_ocp\_status, 212  
 ridge\_slave\_init  
     ridge\_slave.h, 211  
 ridge\_slave\_is\_host\_connected  
     ridge\_slave.h, 211  
 ridge\_slave\_reg\_addr\_t  
     ridge\_slave.h, 210  
 ridge\_slave\_sleep  
     ridge\_slave.h, 212  
 ridge\_slave\_status\_update  
     ridge\_slave.h, 212  
 ridge\_slave\_task  
     ridge\_slave.h, 212  
 ridge\_slave\_update\_ocp\_status  
     ridge\_slave.h, 212  
 ridge\_stat  
     ridge\_reg\_t, 119  
 role\_at\_connect  
     dpm\_status\_t, 64

rp\_cc\_status\_t  
    pd.h, 298

rp\_status  
    pd\_port\_status\_t::PD\_PORT\_STAT, 105

rp\_supported  
    dpm\_status\_t, 64  
    pd\_port\_config\_t, 101

rp\_term\_t  
    pd.h, 299

rsrvd  
    pd\_do\_t::FIXED\_SNK, 69

rsrvd1  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
    pd\_do\_t::RDO\_BAT, 111  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 115  
    pd\_do\_t::RDO\_FIXED\_VAR, 114  
    pd\_do\_t::RDO\_GEN\_GVB, 118  
    pd\_do\_t::RDO\_GEN, 117

rsrvd2  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 112  
    pd\_do\_t::RDO\_BAT, 111  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 116  
    pd\_do\_t::RDO\_FIXED\_VAR, 114  
    pd\_do\_t::RDO\_GEN\_GVB, 119  
    pd\_do\_t::RDO\_GEN, 117

rsvd  
    pd\_do\_t::DP\_STATUS\_VDO, 52  
    pd\_do\_t::STD\_DP\_VDO, 127  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 146

rsvd1  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::BIST\_DO, 45  
    pd\_do\_t::DP\_CONFIG\_VDO, 50  
    pd\_do\_t::PAS\_CBL\_VDO, 79  
    pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
    pd\_do\_t::STD\_AMA\_VDO, 122  
    pd\_do\_t::STD\_CBL\_VDO, 125  
    pd\_do\_t::STD\_VDM\_HDR, 130  
    pd\_do\_t::USTD\_VDM\_HDR, 149  
    pd\_extd\_hdr\_t, 89  
    pd\_hdr\_t::PD\_HDR, 91

rsvd2  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::BIST\_DO, 45  
    pd\_do\_t::DP\_CONFIG\_VDO, 50  
    pd\_do\_t::PAS\_CBL\_VDO, 80  
    pd\_do\_t::STD\_VDM\_HDR, 130  
    ridge\_reg\_t::USBPD\_CMD\_REG, 142

rsvd3  
    pd\_do\_t::PAS\_CBL\_VDO, 80  
    ridge\_reg\_t::USBPD\_CMD\_REG, 143

rsvd4  
    ridge\_reg\_t::USBPD\_CMD\_REG, 143  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 146

SNK\_DETACH\_VBUS\_POLL\_COUNT  
    pd.h, 279

SNK\_MIN\_MAX\_MASK  
    pd.h, 279

SRC\_DRP\_MIN\_DC  
    pd.h, 279

STD\_SVID  
    pd.h, 279

STD\_VDM\_VERSION\_IDX  
    pd.h, 280

STD\_VDM\_VERSION\_REV2  
    pd.h, 280

STD\_VDM\_VERSION\_REV3  
    pd.h, 280

STD\_VDM\_VERSION  
    pd.h, 280

sbu\_switch\_state\_t  
    pdss\_hal.h, 343

scb/i2c.h, 370

scp\_settings\_t, 120  
    debounce, 120  
    retry\_cnt, 120  
    table\_len, 120  
    threshold, 121

scp\_tbl\_offset  
    pd\_port\_config\_t, 101

sel\_conf  
    pd\_do\_t::DP\_CONFIG\_VDO, 50

sense\_res  
    ocp\_settings\_t, 76

seq\_num  
    pd\_do\_t::USTD\_VDM\_HDR, 149

set\_mux  
    alt\_mode\_hw.h, 160

set\_mux\_isolate  
    alt\_mode\_info\_t, 26

setup\_pkt  
    usb\_handle\_t, 137

sign  
    pd\_do\_t::DP\_CONFIG\_VDO, 51

signal  
    pd\_do\_t::STD\_DP\_VDO, 128

sink\_discharge\_off\_cbk\_t  
    pd.h, 284

skip\_scan  
    dpm\_status\_t, 64

slave\_address  
    i2c\_scb\_config\_t, 75

slave\_mask  
    i2c\_scb\_config\_t, 75

sln\_pd\_event\_handler  
    app.h, 192

snk\_cur\_level  
    dpm\_status\_t, 64

snk\_dis\_cbk  
    app\_status\_t, 36

snk\_max\_min  
    dpm\_status\_t, 64

snk\_pdo  
    dpm\_status\_t, 65

snk\_pdo\_cnt  
    pd\_port\_config\_t, 101

snk\_pdo\_count  
     dpm\_status\_t, 65  
 snk\_pdo\_list  
     pd\_port\_config\_t, 101  
 snk\_pdo\_mask  
     dpm\_status\_t, 65  
 snk\_pdo\_max\_min\_current\_pwr  
     pd\_port\_config\_t, 101  
 snk\_period  
     dpm\_status\_t, 65  
 snk\_rdo  
     dpm\_status\_t, 65  
 snk\_rp\_detach\_en  
     dpm\_status\_t, 65  
 snk\_sel  
     chg\_cfg\_params\_t, 48  
 snk\_sel\_pdo  
     dpm\_status\_t, 65  
 snk\_usb\_comm\_en  
     dpm\_status\_t, 65  
     pd\_port\_config\_t, 101  
 snk\_usb\_susp\_en  
     dpm\_status\_t, 65  
     pd\_port\_config\_t, 101  
 sof\_data  
     usb\_handle\_t, 138  
 soft\_rst  
     ridge\_reg\_t::USBPD\_CMD\_REG, 143  
 soln\_no\_vsys\_handler  
     pdss\_hal.h, 369  
 soln\_vsys\_removal\_handler  
     pdss\_hal.h, 370  
 sop  
     pd\_packet\_extd\_t, 93  
     pd\_packet\_t, 94  
 sop\_dp  
     pd\_do\_t::ACT\_CBL\_VDO, 20  
     pd\_do\_t::STD\_CBL\_VDO, 125  
 sop\_state  
     alt\_mode\_info\_t, 26  
 sop\_t  
     pd.h, 299  
 sop\_type  
     vdm\_msg\_info\_t, 153  
 spec\_rev  
     pd\_hdr\_t::PD\_HDR, 91  
 spec\_rev\_cbl  
     dpm\_status\_t, 66  
 spec\_rev\_peer  
     dpm\_status\_t, 66  
 spec\_rev\_sop\_live  
     dpm\_status\_t, 66  
 spec\_rev\_sop\_prime\_live  
     dpm\_status\_t, 66  
 spm\_cfg\_tbl\_offset  
     pd\_port\_config\_t, 102  
 src\_cap\_p  
     dpm\_status\_t, 66  
 src\_cur\_level  
     dpm\_status\_t, 66  
 src\_cur\_level\_live  
     dpm\_status\_t, 66  
 src\_cur\_rdo  
     dpm\_status\_t, 66  
 src\_gen  
     pd\_do\_t, 87  
 src\_last\_rdo  
     dpm\_status\_t, 66  
 src\_pdo  
     dpm\_status\_t, 67  
 src\_pdo\_cnt  
     pd\_port\_config\_t, 102  
 src\_pdo\_count  
     dpm\_status\_t, 67  
 src\_pdo\_list  
     pd\_port\_config\_t, 102  
 src\_pdo\_mask  
     dpm\_status\_t, 67  
 src\_period  
     dpm\_status\_t, 67  
 src\_rdo  
     dpm\_status\_t, 67  
 src\_sel  
     chg\_cfg\_params\_t, 48  
 src\_sel\_pdo  
     dpm\_status\_t, 67  
 ssrx1  
     pd\_do\_t::STD\_AMA\_VDO, 122  
     pd\_do\_t::STD\_CBL\_VDO, 125  
 ssrx2  
     pd\_do\_t::STD\_AMA\_VDO, 122  
     pd\_do\_t::STD\_CBL\_VDO, 125  
 sstx1  
     pd\_do\_t::STD\_AMA\_VDO, 122  
     pd\_do\_t::STD\_CBL\_VDO, 126  
 sstx2  
     pd\_do\_t::STD\_AMA\_VDO, 122  
     pd\_do\_t::STD\_CBL\_VDO, 126  
 st\_ver  
     pd\_do\_t::STD\_VDM\_HDR, 130  
 state  
     bb\_handle\_t, 42  
     cc\_state\_t, 46  
     usb\_handle\_t, 138  
     usb\_i2cm\_t, 139  
 status  
     fw\_img\_status\_t, 72  
     pd\_contract\_info\_t, 83  
     pd\_port\_status\_t, 106  
     usb\_i2cm\_t, 139  
 status.h  
     CCG\_STATUS\_CODE\_OFFSET, 395  
     ccg\_status\_t, 395  
 std\_ama\_vdo  
     pd\_do\_t, 87  
 std\_ama\_vdo\_pd3

pd\_do\_t, 87  
std\_cbl\_vdo  
    pd\_do\_t, 87  
std\_cert\_vdo  
    pd\_do\_t, 87  
std\_dp\_vdo  
    pd\_do\_t, 87  
std\_id\_hdr  
    pd\_do\_t, 87  
std\_prod\_vdo  
    pd\_do\_t, 88  
std\_svid\_res  
    pd\_do\_t, 88  
std\_vdm\_cmd\_t  
    pd.h, 299  
std\_vdm\_cmd\_type\_t  
    pd.h, 300  
std\_vdm\_hdr  
    pd\_do\_t, 88  
std\_vdm\_prod\_t  
    pd.h, 300  
std\_vdm\_ver\_t  
    pd.h, 300  
supply\_type  
    pd\_do\_t::BAT\_SNK, 39  
    pd\_do\_t::BAT\_SRC, 40  
    pd\_do\_t::FIXED\_SNK, 70  
    pd\_do\_t::FIXED\_SRC, 71  
    pd\_do\_t::SRC\_GEN, 121  
    pd\_do\_t::VAR\_SNK, 151  
    pd\_do\_t::VAR\_SRC, 152  
suspend\_check  
    usb\_handle\_t, 138  
svid  
    alt\_mode\_evt\_t::ALT\_MODE\_EVT, 21  
    comp\_tbl\_t, 49  
    pd\_do\_t::STD\_VDM\_HDR, 130  
    pd\_do\_t::USTD\_QC\_4\_0\_HDR, 148  
    pd\_do\_t::USTD\_VDM\_HDR, 149  
svid\_emca\_vdo  
    alt\_mode\_reg\_info\_t, 28  
svid\_n  
    pd\_do\_t::STD\_SVID\_RESP\_VDO, 129  
svid\_n1  
    pd\_do\_t::STD\_SVID\_RESP\_VDO, 129  
svid\_vdm\_length  
    pd\_port\_config\_t, 102  
svid\_vdm\_offset  
    pd\_port\_config\_t, 102  
svid\_vdo  
    alt\_mode\_reg\_info\_t, 28  
swap.h  
    eval\_dr\_swap, 213  
    eval\_pr\_swap, 214  
    eval\_vconn\_swap, 214  
swap\_response  
    dpm\_status\_t, 67  
    pd\_port\_config\_t, 102  
sys\_fw\_metadata\_t, 131  
active\_boot\_app, 132  
boot\_app\_id, 132  
boot\_app\_ver\_status, 132  
boot\_app\_version, 132  
boot\_last\_row, 132  
boot\_seq, 132  
fw\_checksum, 132  
fw\_entry, 133  
fw\_size, 133  
fw\_version, 133  
metadata\_valid, 133  
reserved1, 133  
reserved2, 133  
sys\_fw\_mode\_t  
    system.h, 397  
sys\_get\_bcdDevice\_version  
    system.h, 397  
sys\_get\_boot\_version  
    system.h, 398  
sys\_get\_custom\_info\_addr  
    system.h, 398  
sys\_get\_device\_mode  
    system.h, 398  
sys\_get\_fw\_img1\_start\_addr  
    system.h, 399  
sys\_get\_fw\_img2\_start\_addr  
    system.h, 399  
sys\_get\_img1\_fw\_version  
    system.h, 399  
sys\_get\_img2\_fw\_version  
    system.h, 399  
sys\_get\_recent\_fw\_image  
    system.h, 400  
sys\_get\_silicon\_id  
    system.h, 400  
sys\_hw\_error\_type\_t  
    app.h, 185  
sys\_set\_device\_mode  
    system.h, 400  
system.h  
    get\_silicon\_revision, 397  
    sys\_fw\_mode\_t, 397  
    sys\_get\_bcdDevice\_version, 397  
    sys\_get\_boot\_version, 398  
    sys\_get\_custom\_info\_addr, 398  
    sys\_get\_device\_mode, 398  
    sys\_get\_fw\_img1\_start\_addr, 399  
    sys\_get\_fw\_img2\_start\_addr, 399  
    sys\_get\_img1\_fw\_version, 399  
    sys\_get\_img2\_fw\_version, 399  
    sys\_get\_recent\_fw\_image, 400  
    sys\_get\_silicon\_id, 400  
    sys\_set\_device\_mode, 400  
system/boot.h, 376  
system/ccgx\_api\_desc.h, 380  
system/ccgx\_version.h, 380  
system/flash.h, 381

system/gpio.h, 386  
 system/status.h, 394  
 system/system.h, 396  
 system/timer.h, 401  
 system/utils.h, 406  
 system\_connect\_ovp\_trip  
     hal\_ccgx.h, 323  
 system\_disconnect\_ovp\_trip  
     hal\_ccgx.h, 323  
 system\_init  
     hal\_ccgx.h, 323  
 system\_sleep  
     app.h, 193  
 system\_vbus\_ocp\_dis  
     hal\_ccgx.h, 323  
 system\_vbus\_ocp\_en  
     hal\_ccgx.h, 324  
 system\_vbus\_scp\_dis  
     hal\_ccgx.h, 324  
 system\_vbus\_scp\_en  
     hal\_ccgx.h, 324  
 system\_vconn\_ocp\_dis  
     app.h, 193  
 system\_vconn\_ocp\_en  
     app.h, 193  
  
 TBT\_SVID  
     pd.h, 280  
 TX\_INTERRUPTS  
     pdss\_hal.h, 336  
 TYPEC\_CC\_DEBOUNCE\_TIMER\_PERIOD  
     pd.h, 280  
 TYPEC\_DRP\_TRY\_TIMER\_PERIOD  
     pd.h, 280  
 TYPEC\_ERROR\_RECOVERY\_TIMER\_PERIOD  
     pd.h, 280  
 TYPEC\_FSM\_GENERIC  
     pd.h, 280  
 TYPEC\_FSM\_NONE  
     pd.h, 281  
 TYPEC\_PD3\_RPCCHANGE\_DEBOUNCE\_PERIOD  
     pd.h, 281  
 TYPEC\_PD\_DEBOUNCE\_TIMER\_PERIOD  
     pd.h, 281  
 TYPEC\_RD\_DEBOUNCE\_TIMER\_PERIOD  
     pd.h, 281  
 TYPEC\_SRC\_DETACH\_DEBOUNCE\_PERIOD  
     pd.h, 281  
 table\_checksum  
     pd\_config\_t, 82  
 table\_len  
     bat\_chg\_params\_t, 38  
     bb\_settings\_t, 45  
     chg\_cfg\_params\_t, 48  
     ocp\_settings\_t, 76  
     otp\_settings\_t, 77  
     ovp\_settings\_t, 78  
     pwr\_params\_t, 108  
     scp\_settings\_t, 120  
  
     uvp\_settings\_t, 150  
     vconn\_ocp\_settings\_t, 152  
 table\_sign  
     pd\_config\_t, 82  
 table\_size  
     pd\_config\_t, 82  
 table\_type  
     pd\_config\_t, 83  
 table\_version  
     pd\_config\_t, 83  
 tbt\_cbl\_gen  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 146  
 tbt\_cbl\_spd  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 146  
 tbt\_conn  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 146  
 tbt\_host\_conn  
     ridge\_reg\_t::USBPD\_CMD\_REG, 143  
 tbt\_state\_t  
     intel\_vid.h, 177  
 tbt\_type  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 147  
 temp\_status  
     pd\_power\_status\_t, 107  
 tgt\_id\_header  
     atch\_tgt\_info\_t, 37  
 tgt\_svid  
     atch\_tgt\_info\_t, 37  
 therm\_type  
     otp\_settings\_t, 77  
 threshold  
     ocp\_settings\_t, 76  
     ovp\_settings\_t, 78  
     scp\_settings\_t, 121  
     uvp\_settings\_t, 150  
     vconn\_ocp\_settings\_t, 153  
 threshold2  
     ocp\_settings\_t, 76  
 timeout  
     bb\_handle\_t, 42  
     dpm\_pd\_cmd\_buf\_t, 53  
 timer.h  
     timer\_cb\_t, 402  
     timer\_enter\_sleep, 402  
     timer\_get\_count, 402  
     timer\_get\_multiplier, 403  
     timer\_id\_t, 402  
     timer\_init, 403  
     timer\_is\_running, 403  
     timer\_num\_active, 403  
     timer\_start, 404  
     timer\_start\_woCb, 404  
     timer\_stop, 405  
     timer\_stop\_all, 405  
     timer\_stop\_range, 405  
 timer\_cb\_t  
     timer.h, 402  
 timer\_enter\_sleep

timer.h, 402  
timer\_get\_count  
    timer.h, 402  
timer\_get\_multiplier  
    timer.h, 403  
timer\_id\_t  
    timer.h, 402  
timer\_init  
    timer.h, 403  
timer\_is\_running  
    timer.h, 403  
timer\_num\_active  
    timer.h, 403  
timer\_start  
    timer.h, 404  
timer\_start\_wocb  
    timer.h, 404  
timer\_stop  
    timer.h, 405  
timer\_stop\_all  
    timer.h, 405  
timer\_stop\_range  
    timer.h, 405  
toggle  
    dpm\_status\_t, 67  
    usb\_ep\_handle\_t, 135  
tr\_hpd\_deinit  
    intel\_ridge.h, 175  
tr\_hpd\_init  
    intel\_ridge.h, 175  
tr\_hpd\_sendevt  
    intel\_ridge.h, 175  
tr\_is\_hpd\_change  
    intel\_ridge.h, 176  
try\_src\_en  
    pd\_port\_config\_t, 102  
try\_src\_snk  
    dpm\_status\_t, 68  
try\_src\_snk\_t  
    pd.h, 301  
type  
    bb\_handle\_t, 42  
type\_a\_chg\_tbl\_offset  
    pd\_port\_config\_t, 102  
type\_a\_enable  
    pd\_port\_config\_t, 102  
type\_a\_pwr\_tbl\_offset  
    pd\_port\_config\_t, 103  
typec\_abc  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::PAS\_CBL\_VDO, 80  
    pd\_do\_t::STD\_CBL\_VDO, 126  
typec\_assert\_rd  
    typec\_manager.h, 319  
typec\_assert\_rp  
    typec\_manager.h, 319  
typec\_change\_rp  
    typec\_manager.h, 319  
typec\_deepsleep  
    typec\_manager.h, 320  
typec\_evt  
    dpm\_status\_t, 68  
typec\_fsm  
    typec\_manager.h, 320  
typec\_fsm\_state  
    dpm\_status\_t, 68  
typec\_fsm\_state\_t  
    pd.h, 301  
typec\_init  
    typec\_manager.h, 320  
typec\_is\_busy  
    typec\_manager.h, 320  
typec\_manager.h  
    typec\_assert\_rd, 319  
    typec\_assert\_rp, 319  
    typec\_change\_rp, 319  
    typec\_deepsleep, 320  
    typec\_fsm, 320  
    typec\_init, 320  
    typec\_is\_busy, 320  
    typec\_start, 321  
    typec\_stop, 321  
    typec\_wakeup, 321  
typec\_plug  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::PAS\_CBL\_VDO, 80  
    pd\_do\_t::STD\_CBL\_VDO, 126  
typec\_start  
    typec\_manager.h, 321  
typec\_stop  
    typec\_manager.h, 321  
typec\_wakeup  
    typec\_manager.h, 321  
USB\_REMOTE\_WAKEUP\_TIMER\_PERIOD  
    usb.h, 412  
USB\_SUSPEND\_TIMER\_PERIOD  
    usb.h, 412  
USB\_TARGET\_MASK  
    usbconst.h, 427  
USBARB16  
    usb.h, 425  
USBARB16\_REGS\_T, 140  
USBARB\_REGS\_T, 141  
USBARB  
    usb.h, 425  
USBDEV\_ARB\_EPx\_CFG\_CRC\_BYPASS  
    usb.h, 412  
USBDEV\_ARB\_EPx\_CFG\_DMA\_REQ  
    usb.h, 412  
USBDEV\_ARB\_EPx\_CFG\_IN\_DATA\_RDY  
    usb.h, 412  
USBDEV\_ARB\_EPx\_CFG\_RESET\_PTR  
    usb.h, 412  
USBDEV\_ARB\_RWx\_DR16\_ADDRESS  
    usb.h, 413  
USBDEV\_ARB\_RWx\_DR\_ADDRESS

usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT0\_ADDRESS  
     usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_COUNT\_MSB\_M←  
     ASK  
     usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_TOGGLE  
     usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_VALID  
     usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT1\_ADDRESS  
     usb.h, 413  
 USBDEV\_SIE\_EPx\_CNT1\_DATA\_COUNT\_MASK  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_ACKED\_TXN  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_ADDRESS  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_ERR\_IN\_TXN  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_MODE\_MASK  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_NAK\_INT\_EN  
     usb.h, 414  
 USBDEV\_SIE\_EPx\_CR0\_STALL  
     usb.h, 414  
 USBSIE\_REGS\_T, 147  
 USBSIE  
     usb.h, 425  
 ufp\_asgmt  
     pd\_do\_t::DP\_CONFIG\_VDO, 51  
 ufp\_d\_pin  
     pd\_do\_t::STD\_DP\_VDO, 128  
 unchunk\_sup  
     pd\_do\_t::FIXED\_SRC, 71  
     pd\_do\_t::RDO\_BAT\_GIVEBACK, 113  
     pd\_do\_t::RDO\_BAT, 111  
     pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 116  
     pd\_do\_t::RDO\_FIXED\_VAR, 114  
     pd\_do\_t::RDO\_GEN\_GVB, 119  
     pd\_do\_t::RDO\_GEN, 117  
 unchunk\_sup\_live  
     dpm\_status\_t, 68  
 unchunk\_sup\_peer  
     dpm\_status\_t, 68  
 update\_hpi\_sof\_reset\_timer\_id  
     hpi.h, 242  
 update\_hpi\_soft\_reset\_delay  
     hpi.h, 242  
 usb.h  
     USB\_REMOTE\_WAKEUP\_TIMER\_PERIOD, 412  
     USB\_SUSPEND\_TIMER\_PERIOD, 412  
     USBARB16, 425  
     USBARB, 425  
     USBDEV\_ARB\_EPx\_CFG\_CRC\_BYPASS, 412  
     USBDEV\_ARB\_EPx\_CFG\_DMA\_REQ, 412  
     USBDEV\_ARB\_EPx\_CFG\_IN\_DATA\_RDY, 412  
     USBDEV\_ARB\_EPx\_CFG\_RESET\_PTR, 412  
 USBDEV\_ARB\_RWx\_DR16\_ADDRESS, 413  
 USBDEV\_ARB\_RWx\_DR\_ADDRESS, 413  
 USBDEV\_SIE\_EPx\_CNT0\_ADDRESS, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_COUNT\_MS←  
     B\_MASK, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_TOGGLE, 413  
 USBDEV\_SIE\_EPx\_CNT0\_DATA\_VALID, 413  
 USBDEV\_SIE\_EPx\_CNT1\_ADDRESS, 413  
 USBDEV\_SIE\_EPx\_CNT1\_DATA\_COUNT\_MA←  
     SK, 414  
 USBDEV\_SIE\_EPx\_CR0\_ACKED\_TXN, 414  
 USBDEV\_SIE\_EPx\_CR0\_ADDRESS, 414  
 USBDEV\_SIE\_EPx\_CR0\_ERR\_IN\_TXN, 414  
 USBDEV\_SIE\_EPx\_CR0\_MODE\_MASK, 414  
 USBDEV\_SIE\_EPx\_CR0\_NAK\_INT\_EN, 414  
 USBDEV\_SIE\_EPx\_CR0\_STALL, 414  
 USBSIE, 425  
 usb\_disable, 416  
 usb\_enable, 417  
 usb\_ep0\_send\_recv\_status, 417  
 usb\_ep0\_set\_stall, 417  
 usb\_ep0\_setup\_read, 417  
 usb\_ep0\_setup\_write, 418  
 usb\_ep0\_state\_t, 415  
 usb\_ep0\_wait\_for\_ack, 418  
 usb\_ep\_clear\_stall, 419  
 usb\_ep\_disable, 419  
 usb\_ep\_enable, 419  
 usb\_ep\_flush, 420  
 usb\_ep\_index\_t, 415  
 usb\_ep\_is\_ready, 420  
 usb\_ep\_queue\_read\_single, 421  
 usb\_ep\_read\_single, 421  
 usb\_ep\_send\_zlp, 421  
 usb\_ep\_set\_stall, 422  
 usb\_ep\_write\_single, 422  
 usb\_event\_cb\_t, 414  
 usb\_get\_state, 422  
 usb\_init, 423  
 usb\_intr\_lock, 423  
 usb\_intr\_unlock, 423  
 usb\_is\_idle, 424  
 usb\_remote\_wakeup, 424  
 usb\_setup\_cb\_t, 415  
 usb\_state\_t, 416  
 usb\_task, 424  
 usb\_vbus\_int\_handler, 424  
 usbdev\_ep\_mode\_t, 416  
 usb/usb.h, 409  
 usb/usbconst.h, 425  
 usb2\_0  
     pd\_do\_t::STD\_DP\_VDO, 128  
 usb2\_conn  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 147  
 usb3\_conn  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 147  
 usb3\_speed  
     ridge\_reg\_t::USBPD\_STATUS\_REG, 147

usb\_cfg  
    pd\_do\_t::DP\_STATUS\_VDO, 52

usb\_comm\_cap  
    pd\_do\_t::FIXED\_SNK, 70  
    pd\_do\_t::FIXED\_SRC, 71  
    pd\_do\_t::RDO\_BAT\_GIVEBACK, 113  
    pd\_do\_t::RDO\_BAT, 111  
    pd\_do\_t::RDO\_FIXED\_VAR\_GIVEBACK, 116  
    pd\_do\_t::RDO\_FIXED\_VAR, 114  
    pd\_do\_t::RDO\_GEN\_GVB, 119  
    pd\_do\_t::RDO\_GEN, 117

usb\_config\_t, 133  
    bus\_power, 134  
    class\_rqt\_cb, 134  
    event\_cb, 134  
    fallback\_rqt\_cb, 134  
    get\_dscr\_cb, 134  
    vendor\_rqt\_cb, 134

usb\_configured  
    bb\_handle\_t, 42

usb\_dev  
    pd\_do\_t::STD\_VDM\_ID\_HDR, 131

usb\_dev\_cap\_type\_t  
    usbconst.h, 427

usb\_disable  
    usb.h, 416

usb\_dr  
    ridge\_reg\_t::USBPD\_STATUS\_REG, 147

usb\_dscr\_type\_t  
    usbconst.h, 427

usb\_enable  
    usb.h, 417

usb\_ep0\_send\_recv\_status  
    usb.h, 417

usb\_ep0\_set\_stall  
    usb.h, 417

usb\_ep0\_setup\_read  
    usb.h, 417

usb\_ep0\_setup\_write  
    usb.h, 418

usb\_ep0\_state\_t  
    usb.h, 415

usb\_ep0\_wait\_for\_ack  
    usb.h, 418

usb\_ep\_clear\_stall  
    usb.h, 419

usb\_ep\_disable  
    usb.h, 419

usb\_ep\_enable  
    usb.h, 419

usb\_ep\_flush  
    usb.h, 420

usb\_ep\_handle\_t, 135  
    enabled, 135  
    is\_out, 135  
    toggle, 135

usb\_ep\_index\_t  
    usb.h, 415

usb\_ep\_is\_ready  
    usb.h, 420

usb\_ep\_queue\_read\_single  
    usb.h, 421

usb\_ep\_read\_single  
    usb.h, 421

usb\_ep\_send\_zlp  
    usb.h, 421

usb\_ep\_set\_stall  
    usb.h, 422

usb\_ep\_type\_t  
    usbconst.h, 428

usb\_ep\_write\_single  
    usb.h, 422

usb\_event\_cb\_t  
    usb.h, 414

usb\_feature\_select\_t  
    usbconst.h, 428

usb\_get\_state  
    usb.h, 422

usb\_handle\_t, 135  
    active\_alt\_inf, 136  
    active\_cfg, 136  
    cfg, 136  
    dev\_stat, 136  
    ep0\_buffer, 136  
    ep0\_last, 136  
    ep0\_length, 137  
    ep0\_state, 137  
    ep0\_toggle, 137  
    ep0\_xfer\_cb, 137  
    ep0\_zlp\_rq, 137  
    ep\_handle, 137  
    int\_event, 137  
    prev\_state, 137  
    setup\_pkt, 137  
    sof\_data, 138  
    state, 138  
    suspend\_check, 138

usb\_hid.h  
    hid\_report\_id\_t, 215  
    hid\_rqt\_cmd\_t, 216  
    usb\_hid\_class\_rqt\_handler, 217  
    usb\_hid\_flashing\_rqt, 217  
    usb\_hid\_get\_ep0\_buffer, 217  
    usb\_hid\_get\_inf\_dscr, 218  
    usb\_hid\_get\_report\_dscr, 218  
    usb\_hid\_inf\_ctrl, 219  
    usb\_hid\_set\_fw\_locations, 219  
    usb\_i2cm\_bridge\_ctrl, 219

usb\_hid\_class\_rqt\_handler  
    usb\_hid.h, 217

usb\_hid\_flashing\_rqt  
    usb\_hid.h, 217

usb\_hid\_get\_ep0\_buffer  
    usb\_hid.h, 217

usb\_hid\_get\_inf\_dscr  
    usb\_hid.h, 218

usb\_hid\_get\_report\_dscr  
     usb\_hid.h, 218  
 usb\_hid\_inf\_ctrl  
     usb\_hid.h, 219  
 usb\_hid\_set\_fw\_locations  
     usb\_hid.h, 219  
 usb\_host  
     pd\_do\_t::STD\_VDM\_ID\_HDR, 131  
 usb\_host\_conn  
     ridge\_reg\_t::USBPD\_CMD\_REG, 143  
 usb\_i2cm.h  
     usb\_i2cm\_get\_dscr\_rqt\_handler, 221  
     usb\_i2cm\_get\_ep0\_buffer, 222  
     usb\_i2cm\_inf\_ctrl, 222  
     usb\_i2cm\_is\_idle, 222  
     usb\_i2cm\_task, 222  
     usb\_i2cm\_vdr\_rqt\_t, 221  
     usb\_i2cm\_vendor\_rqt\_handler, 223  
 usb\_i2cm\_bridge\_ctrl  
     usb\_hid.h, 219  
 usb\_i2cm\_get\_dscr\_rqt\_handler  
     usb\_i2cm.h, 221  
 usb\_i2cm\_get\_ep0\_buffer  
     usb\_i2cm.h, 222  
 usb\_i2cm\_inf\_ctrl  
     usb\_i2cm.h, 222  
 usb\_i2cm\_is\_idle  
     usb\_i2cm.h, 222  
 usb\_i2cm\_mode  
     bb\_handle\_t, 42  
 usb\_i2cm\_t, 138  
     count, 138  
     ctrl, 139  
     length, 139  
     preamble, 139  
     preamble\_state, 139  
     state, 139  
     status, 139  
 usb\_i2cm\_task  
     usb\_i2cm.h, 222  
 usb\_i2cm\_vdr\_rqt\_t  
     usb\_i2cm.h, 221  
 usb\_i2cm\_vendor\_rqt\_handler  
     usb\_i2cm.h, 223  
 usb\_init  
     usb.h, 423  
 usb\_intr\_lock  
     usb.h, 423  
 usb\_intr\_unlock  
     usb.h, 423  
 usb\_is\_idle  
     usb.h, 424  
 usb\_pid  
     pd\_do\_t::STD\_PROD\_VDO, 128  
 usb\_port  
     bb\_handle\_t, 42  
 usb\_remote\_wakeup  
     usb.h, 424

usb\_setup\_cb\_t  
     usb.h, 415  
 usb\_setup\_cmd\_t  
     usbconst.h, 428  
 usb\_setup\_pkt\_t, 139  
     attrib, 140  
     cmd, 140  
     index, 140  
     length, 140  
     value, 140  
 usb\_ss\_sup  
     pd\_do\_t::ACT\_CBL\_VDO, 20  
     pd\_do\_t::PAS\_CBL\_VDO, 80  
     pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
     pd\_do\_t::STD\_AMA\_VDO, 123  
     pd\_do\_t::STD\_CBL\_VDO, 126  
 usb\_state\_t  
     usb.h, 416  
 usb\_suspend\_sup  
     pd\_do\_t::FIXED\_SRC, 71  
 usb\_task  
     usb.h, 424  
 usb\_vbus\_int\_handler  
     usb.h, 424  
 usb\_vid  
     pd\_do\_t::STD\_VDM\_ID\_HDR, 131  
 usb\_xid  
     pd\_do\_t::STD\_CERT\_VDO, 127  
 usbconst.h  
     USB\_TARGET\_MASK, 427  
     usb\_dev\_cap\_type\_t, 427  
     usb\_dscr\_type\_t, 427  
     usb\_ep\_type\_t, 428  
     usb\_feature\_select\_t, 428  
     usb\_setup\_cmd\_t, 428  
 usbdev\_ep\_mode\_t  
     usb.h, 416  
 usbpd\_cmd\_reg  
     ridge\_reg\_t, 120  
 ustd\_qc\_4\_0\_hdr  
     pd\_do\_t, 88  
 ustd\_vdm\_hdr  
     pd\_do\_t, 88  
 utils.h  
     crc16, 407  
     MAKE\_DWORD, 407  
     mem\_calculate\_byte\_checksum, 408  
     mem\_calculate\_dword\_checksum, 408  
     mem\_calculate\_word\_checksum, 408  
     mem\_copy\_word, 409  
     REV\_BYTE\_ORDER, 407  
 uvdm.h  
     uvdm\_cmd\_opcode\_t, 225  
     uvdm\_response\_state\_t, 225  
 uvdm\_cmd\_opcode\_t  
     uvdm.h, 225  
 uvdm\_response\_state\_t  
     uvdm.h, 225

uvdm\_supp  
    alt\_mode\_info\_t, 27  
uvp\_settings\_t, 150  
    debounce, 150  
    retry\_cnt, 150  
    table\_len, 150  
    threshold, 150  
uvp\_tbl\_offset  
    pd\_port\_config\_t, 103  
  
VBUS\_OCP\_GPIO\_ACTIVE\_HIGH  
    pdss\_hal.h, 337  
VBUS\_OCP\_GPIO\_ACTIVE\_LOW  
    pdss\_hal.h, 337  
VBUS\_OCP\_MODE\_EXT  
    pdss\_hal.h, 337  
VBUS\_OCP\_MODE\_INT\_AUTOCTRL  
    pdss\_hal.h, 337  
VBUS\_OCP\_MODE\_INT\_SW\_DB  
    pdss\_hal.h, 337  
VBUS\_OCP\_MODE\_INT  
    pdss\_hal.h, 337  
VBUS\_OCP\_MODE\_POLLING  
    pdss\_hal.h, 337  
VDM\_HEADER\_IDX  
    pd.h, 281  
VSAFE\_0V\_HARD\_RESET  
    pd.h, 281  
VSAFE\_0V\_PR\_SWAP\_SNK\_SRC  
    pd.h, 281  
VSAFE\_0V  
    pd.h, 281  
VSAFE\_12V  
    pd.h, 282  
VSAFE\_13V  
    pd.h, 282  
VSAFE\_15V  
    pd.h, 282  
VSAFE\_19V  
    pd.h, 282  
VSAFE\_20V  
    pd.h, 282  
VSAFE\_3\_6V  
    pd.h, 282  
VSAFE\_5V  
    pd.h, 282  
VSAFE\_9V  
    pd.h, 282  
val  
    alt\_mode\_evt\_t, 23  
    alt\_mode\_hw\_evt\_t, 24  
    fw\_img\_status\_t, 72  
    pd\_do\_t, 88  
    pd\_extd\_hdr\_t, 90  
    pd\_hdr\_t, 92  
    pd\_port\_status\_t, 106  
    ridge\_reg\_t, 120  
value  
    usb\_setup\_pkt\_t, 140  
  
var\_snk  
    pd\_do\_t, 88  
var\_src  
    pd\_do\_t, 88  
vbatt\_cutoff\_volt  
    bat\_chg\_params\_t, 38  
vbatt\_dischg\_en\_volt  
    bat\_chg\_params\_t, 38  
vbatt\_max\_cur  
    bat\_chg\_params\_t, 38  
vbatt\_max\_volt  
    bat\_chg\_params\_t, 39  
vbus\_cf\_cbk\_t  
    pdss\_hal.h, 339  
vbus\_cur  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::PAS\_CBL\_VDO, 80  
    pd\_do\_t::STD\_CBL\_VDO, 126  
vbus\_dflt\_volt  
    pwr\_params\_t, 109  
vbus\_discharge\_off  
    app.h, 194  
    app\_cbk\_t, 31  
vbus\_discharge\_on  
    app.h, 194  
    app\_cbk\_t, 31  
vbus\_get\_value  
    app.h, 194  
    app\_cbk\_t, 31  
vbus\_is\_present  
    app.h, 195  
    app\_cbk\_t, 32  
vbus\_max\_volt  
    pwr\_params\_t, 109  
vbus\_min\_volt  
    pwr\_params\_t, 109  
vbus\_ocp\_handler  
    hal\_ccgx.h, 325  
vbus\_ovp\_mode\_t  
    pdss\_hal.h, 343  
vbus\_req  
    pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
    pd\_do\_t::STD\_AMA\_VDO, 123  
vbus\_scp\_handler  
    hal\_ccgx.h, 325  
vbus\_thru\_cbl  
    pd\_do\_t::ACT\_CBL\_VDO, 20  
    pd\_do\_t::STD\_CBL\_VDO, 126  
vbus\_uvp\_mode\_t  
    pdss\_hal.h, 344  
vcon\_pwr  
    pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
    pd\_do\_t::STD\_AMA\_VDO, 123  
vcon\_req  
    pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
    pd\_do\_t::STD\_AMA\_VDO, 123  
vconn\_disable  
    app.h, 195

app\_cbk\_t, 32  
 vconn\_enable  
     app.h, 195  
     app\_cbk\_t, 32  
 vconn\_is\_present  
     app.h, 196  
     app\_cbk\_t, 32  
 vconn\_logical  
     dpm\_status\_t, 68  
 vconn\_ocp\_settings\_t, 152  
     debounce, 152  
     reserved\_0, 152  
     table\_len, 152  
     threshold, 153  
 vconn\_ocp\_tbl\_offset  
     pd\_port\_config\_t, 103  
 vconn\_on  
     pd\_port\_status\_t::PD\_PORT\_STAT, 105  
 vconn\_retain  
     dpm\_status\_t, 68  
     pd\_port\_config\_t, 103  
 vconn\_src  
     pd\_port\_status\_t::PD\_PORT\_STAT, 106  
 vdm.h  
     eval\_vdm, 226  
     get\_modes\_vdo\_info, 226  
     vdm\_data\_init, 227  
     vdm\_update\_data, 227  
     vdm\_update\_svid\_resp, 228  
 vdm\_ams\_t  
     pd.h, 301  
 vdm\_data\_init  
     vdm.h, 227  
 vdm\_evt\_t  
     vdm\_task\_mngr.h, 178  
 vdm\_get\_disc\_id\_resp  
     vdm\_task\_mngr.h, 180  
 vdm\_get\_disc\_svid\_resp  
     vdm\_task\_mngr.h, 180  
 vdm\_header  
     alt\_mode\_info\_t, 27  
     vdm\_msg\_info\_t, 153  
 vdm\_msg\_info\_t, 153  
     sop\_type, 153  
     vdm\_header, 153  
 vdo, 153  
     vdo\_numb, 153  
 vdm\_prcs\_failed  
     app\_status\_t, 36  
 vdm\_resp  
     app\_status\_t, 36  
 vdm\_resp\_cbk  
     app\_status\_t, 36  
 vdm\_resp\_cbk\_t  
     pd.h, 284  
 vdm\_resp\_t, 154  
     do\_count, 154  
     no\_resp, 154  
         resp\_buf, 154  
         vdm\_retry\_pending  
             app\_status\_t, 36  
 vdm\_task\_en  
     app\_status\_t, 36  
 vdm\_task\_mngr  
     vdm\_task\_mngr.h, 180  
 vdm\_task\_mngr.h  
     enable\_vdm\_task\_mngr, 179  
     is\_vdm\_task\_idle, 179  
     vdm\_evt\_t, 178  
     vdm\_get\_disc\_id\_resp, 180  
     vdm\_get\_disc\_svid\_resp, 180  
     vdm\_task\_mngr, 180  
     vdm\_task\_mngr\_deinit, 181  
     vdm\_task\_t, 179  
 vdm\_task\_mngr\_alt\_mode\_process  
     alt\_modes\_mngr.h, 169  
 vdm\_task\_mngr\_deinit  
     vdm\_task\_mngr.h, 181  
 vdm\_task\_t  
     vdm\_task\_mngr.h, 179  
 vdm\_type  
     pd\_do\_t::STD\_VDM\_HDR, 130  
     pd\_do\_t::USTD\_QC\_4\_0\_HDR, 148  
     pd\_do\_t::USTD\_VDM\_HDR, 149  
 vdm\_type\_t  
     pd.h, 302  
 vdm\_update\_data  
     vdm.h, 227  
 vdm\_update\_svid\_resp  
     vdm.h, 228  
 vdm\_ver  
     pd\_do\_t::USTD\_VDM\_HDR, 149  
 vdm\_version  
     app\_status\_t, 36  
 vdo  
     alt\_mode\_info\_t, 27  
     vdm\_msg\_info\_t, 153  
 vdo\_max\_numb  
     alt\_mode\_info\_t, 27  
 vdo\_numb  
     alt\_mode\_info\_t, 27  
     vdm\_msg\_info\_t, 153  
 vdo\_version  
     pd\_do\_t::ACT\_CBL\_VDO, 21  
     pd\_do\_t::PAS\_CBL\_VDO, 80  
     pd\_do\_t::STD\_AMA\_VDO\_PD3, 124  
 vendor\_rqt\_cb  
     usb\_config\_t, 134  
 voltage  
     pd\_do\_t::FIXED\_SNK, 70  
     pd\_do\_t::FIXED\_SRC, 71