

libswd
0.0.1

Generated by Doxygen 1.7.1

Sat Mar 12 2011 20:16:31

Contents

1	Serial Wire Debug Open Library.	1
1.1	Introduction	1
1.2	What is this about	1
1.3	How it works	1
1.3.1	SWD Context	1
1.3.2	Functions	2
1.3.3	Commands	2
1.4	Drivers	2
1.5	Example	3
2	Class Index	5
2.1	Class List	5
3	File Index	7
3.1	File List	7
4	Class Documentation	9
4.1	swd_ahbap_t Struct Reference	9
4.1.1	Detailed Description	10
4.2	swd_cmd_t Struct Reference	10
4.2.1	Detailed Description	11
4.3	swd_context_config_t Struct Reference	11
4.3.1	Detailed Description	11
4.4	swd_ctx_t Struct Reference	12
4.4.1	Detailed Description	12
4.5	swd_driver_t Struct Reference	12
4.5.1	Detailed Description	13
4.6	swd_swdp_t Struct Reference	13
4.6.1	Detailed Description	14

4.7	swd_transaction_t Struct Reference	14
4.7.1	Detailed Description	14
5	File Documentation	15
5.1	libswd.c File Reference	15
5.1.1	Detailed Description	19
5.1.2	Function Documentation	19
5.1.2.1	swd_bin32_bitswap	19
5.1.2.2	swd_bin32_parity_even	19
5.1.2.3	swd_bin32_print	20
5.1.2.4	swd_bin32_string	20
5.1.2.5	swd_bin8_bitswap	20
5.1.2.6	swd_bin8_parity_even	20
5.1.2.7	swd_bin8_print	21
5.1.2.8	swd_bin8_string	21
5.1.2.9	swd_bitgen8_request	21
5.1.2.10	swd_bus_read_ack	21
5.1.2.11	swd_bus_read_data_p	22
5.1.2.12	swd_bus_setdir_miso	22
5.1.2.13	swd_bus_setdir_mosi	22
5.1.2.14	swd_bus_write_control	23
5.1.2.15	swd_bus_write_data_ap	23
5.1.2.16	swd_bus_write_data_p	23
5.1.2.17	swd_bus_write_request	24
5.1.2.18	swd_cmd_enqueue	24
5.1.2.19	swd_cmd_enqueue_miso_ack	24
5.1.2.20	swd_cmd_enqueue_miso_data	24
5.1.2.21	swd_cmd_enqueue_miso_data_p	25
5.1.2.22	swd_cmd_enqueue_miso_n_data_p	25
5.1.2.23	swd_cmd_enqueue_miso_nbit	25
5.1.2.24	swd_cmd_enqueue_miso_parity	26
5.1.2.25	swd_cmd_enqueue_miso_trn	26
5.1.2.26	swd_cmd_enqueue_mosi_control	26
5.1.2.27	swd_cmd_enqueue_mosi_dap_reset	26
5.1.2.28	swd_cmd_enqueue_mosi_data	27
5.1.2.29	swd_cmd_enqueue_mosi_data_ap	27
5.1.2.30	swd_cmd_enqueue_mosi_data_p	27

5.1.2.31	swd_cmd_enqueue_mosi_idle	27
5.1.2.32	swd_cmd_enqueue_mosi_jtag2swd	28
5.1.2.33	swd_cmd_enqueue_mosi_n_data_ap	28
5.1.2.34	swd_cmd_enqueue_mosi_n_data_p	28
5.1.2.35	swd_cmd_enqueue_mosi_nbit	28
5.1.2.36	swd_cmd_enqueue_mosi_parity	29
5.1.2.37	swd_cmd_enqueue_mosi_request	29
5.1.2.38	swd_cmd_enqueue_mosi_swd2jtag	29
5.1.2.39	swd_cmd_enqueue_mosi_trn	30
5.1.2.40	swd_cmdq_append	30
5.1.2.41	swd_cmdq_find_root	30
5.1.2.42	swd_cmdq_find_tail	30
5.1.2.43	swd_cmdq_flush	31
5.1.2.44	swd_cmdq_free	31
5.1.2.45	swd_cmdq_free_head	31
5.1.2.46	swd_cmdq_free_tail	31
5.1.2.47	swd_cmdq_init	32
5.1.2.48	swd_dap_detect	32
5.1.2.49	swd_dap_reset	32
5.1.2.50	swd_dap_select	32
5.1.2.51	swd_deinit	33
5.1.2.52	swd_deinit_cmdq	33
5.1.2.53	swd_deinit_ctx	33
5.1.2.54	swd_dp_read_idcode	33
5.1.2.55	swd_drv_transmit	34
5.1.2.56	swd_init	34
5.1.2.57	swd_log	34
5.1.2.58	swd_log_level_set	34
5.2	libswd.h File Reference	35
5.2.1	Detailed Description	45
5.2.2	Define Documentation	45
5.2.2.1	AHB_AP_BD0	45
5.2.2.2	AHB_AP_BD1	45
5.2.2.3	AHB_AP_BD2	45
5.2.2.4	AHB_AP_BD3	45
5.2.2.5	AHB_AP_CONTROLSTATUS	45

5.2.2.6	AHB_AP_DROMT	45
5.2.2.7	AHB_AP_DRW	45
5.2.2.8	AHB_AP_IDR	46
5.2.2.9	AHB_AP_TAR	46
5.2.2.10	SWD_ABORT_BITNUM_DAPABORT	46
5.2.2.11	SWD_CTRLSTAT_BITNUM_ORUNDETECT	46
5.2.2.12	SWD_DATA_MAXBITCOUNT	46
5.2.2.13	SWD_MASKLANE_0	46
5.2.2.14	SWD_REQUEST_START_BITNUM	46
5.2.2.15	SWD_SELECT_BITNUM_CTRLSEL	46
5.2.2.16	SWD_TURNROUND_1_CODE	46
5.2.2.17	SWD_TURNROUND_2_CODE	47
5.2.2.18	SWD_TURNROUND_3_CODE	47
5.2.2.19	SWD_TURNROUND_4_CODE	47
5.2.2.20	SWD_TURNROUND_DEFAULT_VAL	47
5.2.2.21	SWD_TURNROUND_MAX_VAL	47
5.2.2.22	SWD_TURNROUND_MIN_VAL	47
5.2.2.23	SWD_WCR_BITNUM_PRESCALER	47
5.2.2.24	SWD_WCR_BITNUM_TURNROUND	47
5.2.2.25	SWD_WCR_BITNUM_WIREMODE	47
5.2.3	Typedef Documentation	47
5.2.3.1	swd_cmd_t	47
5.2.4	Enumeration Type Documentation	48
5.2.4.1	swd_bool_t	48
5.2.4.2	swd_cmdtype_t	48
5.2.4.3	swd_error_code_t	48
5.2.4.4	swd_loglevel_t	50
5.2.4.5	swd_operation_t	50
5.2.4.6	swd_shiftdir_t	50
5.2.5	Function Documentation	50
5.2.5.1	swd_bin32_bitswap	50
5.2.5.2	swd_bin32_parity_even	51
5.2.5.3	swd_bin32_print	51
5.2.5.4	swd_bin32_string	51
5.2.5.5	swd_bin8_bitswap	52
5.2.5.6	swd_bin8_parity_even	52

5.2.5.7	swd_bin8_print	52
5.2.5.8	swd_bin8_string	52
5.2.5.9	swd_bitgen8_request	53
5.2.5.10	swd_bus_read_ack	53
5.2.5.11	swd_bus_read_data_p	53
5.2.5.12	swd_bus_setdir_miso	54
5.2.5.13	swd_bus_setdir_mosi	54
5.2.5.14	swd_bus_write_control	54
5.2.5.15	swd_bus_write_data_ap	54
5.2.5.16	swd_bus_write_data_p	55
5.2.5.17	swd_bus_write_request	55
5.2.5.18	swd_cmd_enqueue	55
5.2.5.19	swd_cmd_enqueue_miso_ack	56
5.2.5.20	swd_cmd_enqueue_miso_data	56
5.2.5.21	swd_cmd_enqueue_miso_data_p	56
5.2.5.22	swd_cmd_enqueue_miso_n_data_p	56
5.2.5.23	swd_cmd_enqueue_miso_nbit	57
5.2.5.24	swd_cmd_enqueue_miso_parity	57
5.2.5.25	swd_cmd_enqueue_miso_trn	57
5.2.5.26	swd_cmd_enqueue_mosi_control	57
5.2.5.27	swd_cmd_enqueue_mosi_dap_reset	58
5.2.5.28	swd_cmd_enqueue_mosi_data	58
5.2.5.29	swd_cmd_enqueue_mosi_data_ap	58
5.2.5.30	swd_cmd_enqueue_mosi_data_p	59
5.2.5.31	swd_cmd_enqueue_mosi_idle	59
5.2.5.32	swd_cmd_enqueue_mosi_jtag2swd	59
5.2.5.33	swd_cmd_enqueue_mosi_n_data_ap	59
5.2.5.34	swd_cmd_enqueue_mosi_n_data_p	60
5.2.5.35	swd_cmd_enqueue_mosi_nbit	60
5.2.5.36	swd_cmd_enqueue_mosi_parity	60
5.2.5.37	swd_cmd_enqueue_mosi_request	61
5.2.5.38	swd_cmd_enqueue_mosi_swd2jtag	61
5.2.5.39	swd_cmd_enqueue_mosi_trn	61
5.2.5.40	swd_cmdq_append	61
5.2.5.41	swd_cmdq_find_root	62
5.2.5.42	swd_cmdq_find_tail	62

5.2.5.43	swd_cmdq_flush	62
5.2.5.44	swd_cmdq_free	62
5.2.5.45	swd_cmdq_free_head	63
5.2.5.46	swd_cmdq_free_tail	63
5.2.5.47	swd_cmdq_init	63
5.2.5.48	swd_dap_detect	63
5.2.5.49	swd_dap_reset	64
5.2.5.50	swd_dap_select	64
5.2.5.51	swd_deinit	64
5.2.5.52	swd_deinit_cmdq	64
5.2.5.53	swd_deinit_ctx	65
5.2.5.54	swd_dp_read_idcode	65
5.2.5.55	swd_drv_transmit	65
5.2.5.56	swd_init	65
5.2.5.57	swd_log	66
5.2.5.58	swd_log_level_set	66

Chapter 1

Serial Wire Debug Open Library.

1.1 Introduction

LibSWD is an Open-Source framework to deal with with Serial Wire Debug Port in accordance to ADI (Arm Debug Interface, version 5.0 at the moment) specification. It is released under 3-clause BSD license. For more information please visit project website at <http://libswd.sf.net>

1.2 What is this about

Serial Wire Debug is an alternative to JTAG (IEEE1149.1) transport layer for accessing the Debug Access Port in ARM-Cortex based devices. LibSWD provides methods for bitstream generation on the wire using simple but flexible API that can reuse capabilities of existing applications for easier integration. Every bus operation such as control, request, turnaround, acknowledge, data and parity packet is named a "command" represented by a `swd_cmd_t` data type that builds up the queue that later can be flushed into real hardware using standard set of (application-specific) driver functions. This way LibSWD is almost standalone and can be easily integrated into existing utilities for low-level access and only requires in return to define driver bridge that controls the physical interface interconnecting host and target. Drivers and other application-specific functions are "extern" and located in external file crafted for that application and its hardware. LibSWD is therefore best way to make your application SWD aware.

1.3 How it works

1.3.1 SWD Context

The most important data type in LibSWD is `swd_ctx_t` structure, a context that represents logical entity of the swd bus (transport layer between host and target) with all its parameters, configuration and command queue. Context is being created with `swd_init()` function that returns pointer to allocated virgin structure, and it can be destroyed with `swd_deinit()` function taking the pointer as argument. Context can be set only for one interface-target pair, but there might be many different contexts in use if necessary, so amount of devices in use is not limited.

1.3.2 Functions

All functions in general operates on pointer type and returns number of processed elements on success or negative value with `swd_error_code_t` on failure. Functions are grouped by functionality that is denoted by function name prefix (ie. `swd_bin*` are for binary operations, `swd_cmdq*` deals with command queue, `swd_cmd_enqueue*` deals with creating commands and attaching them to queue, `swd_bus*` performs operation on the swd transport system, `swd_drv*` are the interface drivers, etc).

Standard end-users are encouraged to only use high level functions (`swd_bus*`, `swd_dap*`, `swd_dp*`) to perform operations on the swd transport layer and the target's DAP (Debug Access Port) and its components such as DP (Debug Port) and the AP (Access Port). More advanced users however may use low level functions (`swd_cmd*`, `swd_cmdq*`) to group them into new high-level functions that automates some tasks (such as high-level functions does). Functions of type "extern" are the ones to implement in external file by developers that want to incorporate LibSWD into their application. Context structure also has void pointer in the `swd_driver_t` structure that can hold address of the external driver structure to be passed into internal swd drivers (extern `swd_drv*` functions) that wouldn't be accessible otherwise.

1.3.3 Commands

Bus operations are split into "commands" represented by `swd_cmd_t` data type. They form a bidirectional command queue that is part of `swd_ctx_t` structure. Command type, and so its payload, can be one of: control (user defined 8-bit payload), request (according to the standard), ack, data, parity (data and parity are separate commands!), trn, bitbang and idle (equals to control with zero data). Command type is defined by `swd_cmdtype_t` and its code can be negative (for MOSI operations) or positive (for MISO operations) - this way bus direction can be easily calculated by multiplying two operation codes (when the result is negative bus will have to change direction), so the libswd "knows" when to put additional TRN command of proper type between enqueued commands.

Payload is stored within union type and its data can be accessed according to payload name, or simply with `data8 (char)` and `data32 (int)` fields. Payload for write (MOSI) operations is stored on command creation, but payload for read (MISO) operations becomes available only after command is executed by the interface driver. There are 3 methods of accessing read data - flushing the queue into driver then reading queue directly, single stepping queue execution (flush one-by-one) then reading context log of last executed command results (there are separate fields of type `swd_transaction_t` in `swd_ctx_t`'s log structure for read and write operations), or providing a double pointer on command creation to have constant access to its data after execution.

After all commands are enqueued with `swd_cmd_enqueue*` function set, it is time to send them into physical device with `swd_cmdq_flush()` funtion. According to the `swd_operation_t` parameter commands can be flushed one-by-one, all of them, only to the selected command or only after selected command. For low level functions all of these options are available, but for high-level functions only two of them can be used - `SWD_OPERATION_ENQUEUE` (but not send to the driver) and `SWD_OPERATION_EXECUTE` (all unexecuted commands on the queue are executed by the driver sequentially) - that makes it possible to perform bus operations one after another having their result just at function return, or compose more advanced sequences leading to preferred result at execution time. Because high-level functions provide simple and elegant manner to get the operation result, it is advised to use them instead dealing with low-level functions (implementing memory management, data allocation and queue operation) that exist only to make high-level functions possible.

1.4 Drivers

Calling the `swd_cmdq_flush()` function leads to execution of not yet executed commands from the queue (in a manner specified by the operation parameter) on the SWD bus (transport layer between interface and

target, not the bus of the target itself) by `swd_drv_transmit()` function that use application specific "extern" functions defined in external file (ie. `libswd_urjtag.c`) to operate on a real hardware using drivers from existing application. LibSWD use only `swd_drv_{mosi,miso}_{8,32}` (separate for 8-bit char and 32-bit int data cast type) and `swd_drv_{mosi,miso}_trn` functions to interact with drivers, so it is possible to easily reuse low-level and high-level devices for communications, as they have all information necessary to perform exact actions - number of bits, payload, command type, shift direction and bus direction. It is even possible to send raw bytes on the bus (control command) or bitbang the bus (bitbang command) if necessary. MOSI (Master Output Slave Input) and MISO (Master Input Slave Output) was used to clearly distinguish transfer direction (from master-interface to target-slave), as opposed to ambiguous read/write statements, so after `swd_drv_mosi_trn()` master should have its buffers set to output and target inputs active. Drivers, as most of the LibSWD functions, works on data pointers instead data copy and returns number of elements processed (bits in this case) or negative error code on failure.

1.5 Example

```
#include <libswd/libswd.h>
int main() {
    swd_ctx_t *swdctx;
    int res, idcode;
    swdctx=swd_init();
    if (swdctx==NULL) return -1;
    //we might need to pass external driver structure to swd_drv* functions
    //swdctx->driver->device=...
    res=swd_dap_detect(swdctx, SWD_OPERATION_EXECUTE, &idcode);
    if (res<0){
        printf("ERROR: %s\n", swd_error_string(res));
        return res;
    } else printf("IDCODE: 0x%X\n", idcode);
    swd_deinit(swdctx);
    return idcode;
}
```


Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

swd_ahbap_t (Most actual Advanced High Bandwidth Access Peripheral Bus Registers)	9
swd_cmd_t (SWD Command Element Structure)	10
swd_context_config_t (Context configuration structure)	11
swd_ctx_t (SWD Context Structure definition)	12
swd_driver_t (Interface Driver structure)	12
swd_swdp_t (Most actual Serial Wire Debug Port Registers)	13
swd_transaction_t (Most actual SWD bus transaction/packet data)	14

Chapter 3

File Index

3.1 File List

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

libswd.c	15
libswd.h	35

Chapter 4

Class Documentation

4.1 swd_ahbap_t Struct Reference

Most actual Advanced High Bandwidth Access Peripheral Bus Registers.

```
#include <libswd.h>
```

Public Attributes

- char [ack](#)
Last known state of ACK response.
- int [controlstatus](#)
Last known CONTROLSTATUS register value.
- int [tar](#)
Last known TAR register value.
- int [drw](#)
Last known DRW register value.
- int [bd0](#)
Last known BD0 register value.
- int [bd1](#)
Last known BD1 register value.
- int [bd2](#)
Last known BD2 register value.
- int [bd3](#)
Last known BD3 register value.
- int [dromt](#)
Last known DROMT register value.

- int [idr](#)

Last known IDR register value.

4.1.1 Detailed Description

Most actual Advanced High Bandwidth Access Peripheral Bus Registers.

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

- [libswd.h](#)

4.2 swd_cmd_t Struct Reference

SWD Command Element Structure.

```
#include <libswd.h>
```

Public Attributes

- union {
 - char [TRNnMOSI](#)
Holds/sets bus direction: MOSI when zero, MISO for others.
 - char [request](#)
Request header data.
 - char [ack](#)
Acknowledge response from target.
 - int [misodata](#)
Data read from target (MISO).
 - int [mosidata](#)
Data written to target (MOSI).
 - int [data32](#)
Holds "int" data type for inspection.
 - char [misobit](#)
Single bit read from target (bit-per-char).
 - char [mosibit](#)
Single bit written to target (bit-per-char).
 - char [parity](#)
Parity bit for data payload.
 - char [control](#)
Control transfer data (one byte).
 - char [data8](#)
Holds "char" data type for inspection.
};
- char [bits](#)
Payload bit count == clk pulses on the bus.
- [swd_cmdtype_t](#) [cmdtype](#)
Command type as defined by swd_cmdtype_t.

- char [done](#)
Non-zero if operation already executed.
- struct [swd_cmd_t](#) * [prev](#)
Pointer to the previous command.
- struct [swd_cmd_t](#) * [next](#)
Pointer to the next command.

4.2.1 Detailed Description

SWD Command Element Structure. In libswd each operation is split into separate commands (request, trn, ack, data, parity) that can be appended to the command queue and later executed. This organization allows better granularity for tracing bugs and makes possible to compose complete bus/target operations made of simple commands.

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

- [libswd.h](#)

4.3 swd_context_config_t Struct Reference

Context configuration structure.

```
#include <libswd.h>
```

Public Attributes

- char [initialized](#)
Context must be initialized prior use.
- char [trnlen](#)
How many CLK cycles will TRN use.
- int [maxcmdqlen](#)
How long command queue can be.
- [swd_loglevel_t](#) [loglevel](#)
Holds Logging Level setting.

4.3.1 Detailed Description

Context configuration structure.

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

- [libswd.h](#)

4.4 swd_ctx_t Struct Reference

SWD Context Structure definition.

```
#include <libswd.h>
```

Public Attributes

- [swd_cmd_t * cmdq](#)
Command queue, stores all bus operations.
- [swd_context_config_t config](#)
Target specific configuration.
- [swd_driver_t * driver](#)
Pointer to the interface driver structure.
- struct {
 - [swd_swdp_t dp_read](#)
Last known read from the SW-DP registers.
 - [swd_swdp_t dp_write](#)
Last known write to the SW-DP registers.
 - [swd_ahbap_t ap_read](#)
Last known read from AHB-AP registers.
 - [swd_ahbap_t ap_write](#)
Last known write to the AHB-AP registers.
 - [swd_transaction_t read](#)
 - [swd_transaction_t write](#)
- } **log**

4.4.1 Detailed Description

SWD Context Structure definition. It stores all the information about the library, drivers and interface configuration, target status along with DAP/AHBAP data/instruction internal registers, and the command queue. Bus operations are stored on the command queue. There may be more than one context in use by a host software, each one for single interface-target pair. Most of the target operations made with libswd are required to pass [swd_ctx_t](#) pointer structure that also remembers last known state of the target's internal registers.

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

- [libswd.h](#)

4.5 swd_driver_t Struct Reference

Interface Driver structure.

```
#include <libswd.h>
```

Public Attributes

- void * **device**

4.5.1 Detailed Description

Interface Driver structure. It holds pointer to the driver structure that keeps driver information necessary to work with the physical interface.

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

- [libswd.h](#)

4.6 swd_swdp_t Struct Reference

Most actual Serial Wire Debug Port Registers.

```
#include <libswd.h>
```

Public Attributes

- char [ack](#)
Last known state of ACK response.
- char [parity](#)
Parity bit of the data transfer.
- int [idcode](#)
Target's IDCODE register value.
- int [abort](#)
Last known ABORT register value.
- int [ctrlstat](#)
Last known CTRLSTAT register value.
- int [wcr](#)
Last known WCR register value.
- int [select](#)
Last known SELECT register value.
- int [rdbuf](#)
Last known RDBUF register (payload data) value.

4.6.1 Detailed Description

Most actual Serial Wire Debug Port Registers.

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

- [libswd.h](#)

4.7 swd_transaction_t Struct Reference

Most actual SWD bus transaction/packet data.

```
#include <libswd.h>
```

Public Attributes

- char [request](#)
Last known request on the bus.
- char [ack](#)
Last known ack on the bus.
- int [data](#)
Last known data on the bus.
- int [control](#)
Last known control data on the bus.
- char [parity](#)
Last known parity on the bus.

4.7.1 Detailed Description

Most actual SWD bus transaction/packet data.

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

- [libswd.h](#)

Chapter 5

File Documentation

5.1 libswd.c File Reference

```
#include <libswd/libswd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
```

Functions

- int [swd_bin8_parity_even](#) (char *data, char *parity)
Data parity calculator, calculates even parity on char type.
- int [swd_bin32_parity_even](#) (int *data, char *parity)
Data parity calculator, calculates even parity on integer type.
- int [swd_bin8_print](#) (char *data)
Prints binary data of a char value on the screen.
- int [swd_bin32_print](#) (int *data)
Prints binary data of an integer value on the screen.
- char * [swd_bin8_string](#) (char *data)
Generates string containing binary data of a char value.
- char * [swd_bin32_string](#) (int *data)
Generates string containing binary data of an integer value.
- int [swd_bin8_bitswap](#) (unsigned char *buffer, int bitcount)
*Bit swap helper function that reverse bit order in char *buffer.*
- int [swd_bin32_bitswap](#) (unsigned int *buffer, int bitcount)

*Bit swap helper function that reverse bit order in int *buffer.*

- `int swd_cmdq_init (swd_cmd_t *cmdq)`
Initialize new queue element in memory that becomes a queue root.
- `swd_cmd_t * swd_cmdq_find_root (swd_cmd_t *cmdq)`
Find queue root (first element).
- `swd_cmd_t * swd_cmdq_find_tail (swd_cmd_t *cmdq)`
Find queue tail (last element).
- `int swd_cmdq_append (swd_cmd_t *cmdq, swd_cmd_t *cmd)`
*Append element pointed by *cmd at the end of the quque pointed by *cmdq.*
- `int swd_cmdq_free (swd_cmd_t *cmdq)`
*Free queue pointed by *cmdq element.*
- `int swd_cmdq_free_head (swd_cmd_t *cmdq)`
*Free queue head up to *cmdq element.*
- `int swd_cmdq_free_tail (swd_cmd_t *cmdq)`
*Free queue tail starting after *cmdq element.*
- `int swd_cmd_enqueue (swd_ctx_t *swdctx, swd_cmd_t *cmd)`
Append selected command to a context's command queue.
- `int swd_cmd_enqueue_mosi_request (swd_ctx_t *swdctx, char *request)`
Appends command queue with SWD Request packet header.
- `int swd_cmd_enqueue_mosi_trn (swd_ctx_t *swdctx)`
Append command queue with Turnaround activating MOSI mode.
- `int swd_cmd_enqueue_miso_trn (swd_ctx_t *swdctx)`
Append command queue with Turnaround activating MISO mode.
- `int swd_cmd_enqueue_miso_nbit (swd_ctx_t *swdctx, char **data, int count)`
Append command queue with bus binary read bit-by-bit operation.
- `int swd_cmd_enqueue_mosi_nbit (swd_ctx_t *swdctx, char *data, int count)`
Append command queue with bus binary write bit-by-bit operation.
- `int swd_cmd_enqueue_mosi_parity (swd_ctx_t *swdctx, char *parity)`
Append command queue with parity bit write.
- `int swd_cmd_enqueue_miso_parity (swd_ctx_t *swdctx, char **parity)`
Append command queue with parity bit read.
- `int swd_cmd_enqueue_miso_data (swd_ctx_t *swdctx, int **data)`
Append command queue with data read.

- `int swd_cmd_enqueue_miso_data_p (swd_ctx_t *swdctx, int **data, char **parity)`
Append command queue with data and parity read.
- `int swd_cmd_enqueue_miso_n_data_p (swd_ctx_t *swdctx, int **data, char **parity, int count)`
Append command queue with series of data and parity read.
- `int swd_cmd_enqueue_mosi_data (swd_ctx_t *swdctx, int *data)`
Append command queue with data and parity write.
- `int swd_cmd_enqueue_mosi_data_ap (swd_ctx_t *swdctx, int *data)`
Append command queue with data and automatic parity write.
- `int swd_cmd_enqueue_mosi_data_p (swd_ctx_t *swdctx, int *data, char *parity)`
Append command queue with data and provided parity write.
- `int swd_cmd_enqueue_mosi_n_data_ap (swd_ctx_t *swdctx, int **data, int count)`
Append command queue with series of data and automatic parity writes.
- `int swd_cmd_enqueue_mosi_n_data_p (swd_ctx_t *swdctx, int **data, char **parity, int count)`
Append command queue with series of data and provided parity writes.
- `int swd_cmd_enqueue_miso_ack (swd_ctx_t *swdctx, char **ack)`
Append queue with ACK read.
- `int swd_cmd_enqueue_mosi_control (swd_ctx_t *swdctx, char *ctlmsg, int len)`
Append command queue with len-octet size control sequence.
- `int swd_cmd_enqueue_mosi_dap_reset (swd_ctx_t *swdctx)`
Append command queue with SW-DP-RESET sequence.
- `int swd_cmd_enqueue_mosi_idle (swd_ctx_t *swdctx)`
Append command queue with idle sequence.
- `int swd_cmd_enqueue_mosi_jtag2swd (swd_ctx_t *swdctx)`
Append command queue with JTAG-TO-SWD DAP-switch sequence.
- `int swd_cmd_enqueue_mosi_swd2jtag (swd_ctx_t *swdctx)`
Append command queue with SWD-TO-JTAG DAP-switch sequence.
- `char * swd_cmd_string_cmdtype (swd_cmd_t *cmd)`
- `int swd_bitgen8_request (swd_ctx_t *swdctx, char *APnDP, char *RnW, char *addr, char *request)`
Generate 8-bit SWD-REQUEST packet contents with provided parameters.
- `int swd_drv_mosi_8 (swd_ctx_t *swdctx, swd_cmd_t *cmd, char *data, int bits, int nLSBfirst)`
- `int swd_drv_mosi_32 (swd_ctx_t *swdctx, swd_cmd_t *cmd, int *data, int bits, int nLSBfirst)`
- `int swd_drv_miso_8 (swd_ctx_t *swdctx, swd_cmd_t *cmd, char *data, int bits, int nLSBfirst)`
- `int swd_drv_miso_32 (swd_ctx_t *swdctx, swd_cmd_t *cmd, int *data, int bits, int nLSBfirst)`
- `int swd_drv_mosi_trn (swd_ctx_t *swdctx, int bits)`
- `int swd_drv_miso_trn (swd_ctx_t *swdctx, int bits)`

- `int swd_drv_transmit (swd_ctx_t *swdctx, swd_cmd_t *cmd)`
Transmit selected command from the command queue to the interface driver.
- `int swd_cmdq_flush (swd_ctx_t *swdctx, swd_operation_t operation)`
Flush command queue contents into interface driver.
- `int swd_bus_setdir_mosi (swd_ctx_t *swdctx)`
Append command queue with TRN WRITE/MOSI, if previous command was READ/MISO.
- `int swd_bus_setdir_miso (swd_ctx_t *swdctx)`
Append command queue with TRN READ/MISO, if previous command was WRITE/MOSI.
- `int swd_bus_write_request (swd_ctx_t *swdctx, swd_operation_t operation, char *APnDP, char *RnW, char *addr)`
Perform Request.
- `int swd_bus_read_ack (swd_ctx_t *swdctx, swd_operation_t operation, char **ack)`
*Perform ACK read into *ack and verify received data.*
- `int swd_bus_write_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int *data, char *parity)`
Perform (MOSI) data write with provided parity value.
- `int swd_bus_write_data_ap (swd_ctx_t *swdctx, swd_operation_t operation, int *data)`
Perform (MOSI) data write with automatic parity calculation.
- `int swd_bus_read_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int **data, char **parity)`
Perform (MISO) data read.
- `int swd_bus_write_control (swd_ctx_t *swdctx, swd_operation_t operation, char *ctlmsg, int len)`
Write CONTROL byte to the Target's DAP.
- `int swd_dap_reset (swd_ctx_t *swdctx, swd_operation_t operation)`
Debug Access Port Reset sends 50 CLK with TMS high that brings both SW-DP and JTAG-DP into reset state.
- `int swd_dap_select (swd_ctx_t *swdctx, swd_operation_t operation)`
Activate SW-DP by sending out RESET and JTAG-TO-SWD sequence on SWDIOTMS line.
- `int swd_dp_read_idcode (swd_ctx_t *swdctx, swd_operation_t operation, int **idcode)`
Macro: Read out IDCODE register and return its value on function return.
- `int swd_dap_detect (swd_ctx_t *swdctx, swd_operation_t operation, int **idcode)`
Macro: Reset target DAP, select SW-DP, read out IDCODE.
- `int swd_log (swd_ctx_t *swdctx, swd_loglevel_t loglevel, char *msg,...)`
Put a message into swd context log at specified verbosity level.
- `int swd_log_level_set (swd_ctx_t *swdctx, swd_loglevel_t loglevel)`
Change log level to increase or decrease verbosity level.

- char * **swd_error_string** (swd_error_code_t error)
- swd_ctx_t * **swd_init** (void)
LibSWD initialization routine.
- int **swd_deinit_ctx** (swd_ctx_t *swdctx)
De-initialize selected swd context and free its memory.
- int **swd_deinit_cmdq** (swd_ctx_t *swdctx)
De-initialize command queue and free its memory on selected swd context.
- int **swd_deinit** (swd_ctx_t *swdctx)
De-initialize selected swd context and its command queue.

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 int swd_bin32_bitswap (unsigned int * *buffer*, int *bitcount*)

Bit swap helper function that reverse bit order in int *buffer.

Most Significant Bit becomes Least Significant Bit. It is possible to swap only n-bits from int (32-bit) *buffer.

Parameters

**buffer* unsigned char (32-bit) data pointer.

bitcount how many bits to swap.

Returns

swapped bit count (positive) or error code (negative).

5.1.2.2 int swd_bin32_parity_even (int * *data*, char * *parity*)

Data parity calculator, calculates even parity on integer type.

Parameters

**data* source data pointer.

**parity* resulting data pointer.

Returns

negative value on error, 0 or 1 as parity result.

5.1.2.3 int swd_bin32_print (int * *data*)

Prints binary data of an integer value on the screen.

Parameters

**data* source data pointer.

Returns

number of characters printed.

5.1.2.4 char* swd_bin32_string (int * *data*)

Generates string containing binary data of an integer value.

Parameters

**data* source data pointer.

Returns

pointer to the resulting string.

5.1.2.5 int swd_bin8_bitswap (unsigned char * *buffer*, int *bitcount*)

Bit swap helper function that reverse bit order in char *buffer.

Most Significant Bit becomes Least Significant Bit. It is possible to swap only n-bits from char (8-bit) *buffer.

Parameters

**buffer* unsigned char (8-bit) data pointer.

bitcount how many bits to swap.

Returns

swapped bit count (positive) or error code (negative).

5.1.2.6 int swd_bin8_parity_even (char * *data*, char * *parity*)

Data parity calculator, calculates even parity on char type.

Some comments on the function behavior.

Parameters

**data* source data pointer.

**parity* resulting data pointer.

Returns

negative value on error, 0 or 1 as parity result.

5.1.2.7 int swd_bin8_print (char * *data*)

Prints binary data of a char value on the screen.

Parameters

**data* source data pointer.

Returns

number of characters printed.

5.1.2.8 char* swd_bin8_string (char * *data*)

Generates string containing binary data of a char value.

Parameters

**data* source data pointer.

Returns

pointer to the resulting string.

5.1.2.9 int swd_bitgen8_request (swd_ctx_t * *swdctx*, char * *APnDP*, char * *RnW*, char * *addr*, char * *request*)

Generate 8-bit SWD-REQUEST packet contents with provided parameters.

Note that parity bit value is calculated automatically.

Parameters

**swdctx* swd context pointer.

**APnDP* AccessPort (high) or DebugPort (low) access type pointer.

**RnW* Read (high) or Write (low) operation type pointer.

**addr* target register address value pointer.

**request* pointer where to store resulting packet.

Returns

number of generated packets (1), or SWD_ERROR_CODE on failure.

5.1.2.10 int swd_bus_read_ack (swd_ctx_t * *swdctx*, swd_operation_t *operation*, char ** *ack*)

Perform ACK read into **ack* and verify received data.

Parameters

**swdctx* swd context pointer.

operation type of action to perform with generated request.

**ack* pointer to the result location.

Returns

number of commands processed, or SWD_ERROR_CODE on failure.

5.1.2.11 `int swd_bus_read_data_p (swd_ctx_t * swdctx, swd_operation_t operation, int ** data, char ** parity)`

Perform (MISO) data read.

Parameters

**swdctx* swd context pointer.

operation type of action to perform on generated command.

**data* payload value pointer.

**parity* payload parity value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.1.2.12 `int swd_bus_setdir_miso (swd_ctx_t * swdctx)`

Append command queue with TRN READ/MISO, if previous command was WRITE/MOSI.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.13 `int swd_bus_setdir_mosi (swd_ctx_t * swdctx)`

Append command queue with TRN WRITE/MOSI, if previous command was READ/MISO.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.14 `int swd_bus_write_control (swd_ctx_t * swdctx, swd_operation_t operation, char * ctlmsg, int len)`

Write CONTROL byte to the Target's DAP.

Parameters

**swdctx* swd context.

operation can be SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE.

**ctlmsg* byte/char array that contains control payload.

len number of bytes in the **ctlmsg* to send.

Returns

number of bytes sent or SWD_ERROR_CODE on failure.

5.1.2.15 `int swd_bus_write_data_ap (swd_ctx_t * swdctx, swd_operation_t operation, int * data)`

Perform (MOSI) data write with automatic parity calculation.

Parameters

**swdctx* swd context pointer.

operation type of action to perform on generated command.

**data* payload value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.1.2.16 `int swd_bus_write_data_p (swd_ctx_t * swdctx, swd_operation_t operation, int * data, char * parity)`

Perform (MOSI) data write with provided parity value.

Parameters

**swdctx* swd context pointer.

operation type of action to perform on generated command.

**data* payload value pointer.

**parity* payload parity value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.1.2.17 `int swd_bus_write_request (swd_ctx_t * swdctx, swd_operation_t operation, char * APnDP, char * RnW, char * addr)`

Perform Request.

Parameters

- *swdctx* swd context pointer.
- operation* type of action to perform with generated request.
- *APnDP* AccessPort (high) or DebugPort (low) access value pointer.
- *RnW* Read (high) or Write (low) access value pointer.
- *addr* target register address value pointer.

Returns

number of commands processed, or SWD_ERROR_CODE on failure.

5.1.2.18 `int swd_cmd_enqueue (swd_ctx_t * swdctx, swd_cmd_t * cmd)`

Append selected command to a context's command queue.

Parameters

- *swdctx* swd context pointer containing the command queue.
- *cmd* command to be appended to the context's command queue.

Returns

number of elements appended or SWD_ERROR_CODE on failure.

5.1.2.19 `int swd_cmd_enqueue_miso_ack (swd_ctx_t * swdctx, char ** ack)`

Append queue with ACK read.

Parameters

- *swdctx* swd context pointer.
- *ack* packet value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.20 `int swd_cmd_enqueue_miso_data (swd_ctx_t * swdctx, int ** data)`

Append command queue with data read.

Parameters

- *swdctx* swd context pointer.
- *data* data pointer.

Returns

of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.21 int swd_cmd_enqueue_miso_data_p (swd_ctx_t * swdctx, int ** data, char ** parity)

Append command queue with data and parity read.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

**parity* parity value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.1.2.22 int swd_cmd_enqueue_miso_n_data_p (swd_ctx_t * swdctx, int ** data, char ** parity, int count)

Append command queue with series of data and parity read.

Parameters

**swdctx* swd context pointer.

***data* data value array pointer.

***parity* parity value array pointer.

count number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.1.2.23 int swd_cmd_enqueue_miso_nbit (swd_ctx_t * swdctx, char ** data, int count)

Append command queue with bus binary read bit-by-bit operation.

This function will append command to the queue for each bit, and store one bit into single char array element, so read is not constrained to 8 bits. On error memory is released and appropriate error code is returned. Important: Memory pointed by *data must be allocated prior call!

Parameters

**swdctx* swd context pointer.

***data* allocated data array to write result into.

count number of bits to read (also the **data size).

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.1.2.24 int swd_cmd_enqueue_miso_parity (swd_ctx_t * swdctx, char ** parity)

Append command queue with parity bit read.

Parameters

**swdctx* swd context pointer.

**parity* parity value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.25 int swd_cmd_enqueue_miso_trn (swd_ctx_t * swdctx)

Append command queue with Turnaround activating MISO mode.

Parameters

**swdctx* swd context pointer.

Returns

return number of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.26 int swd_cmd_enqueue_mosi_control (swd_ctx_t * swdctx, char * ctlmsg, int len)

Append command queue with len-octet size control sequence.

This control sequence can be used for instance to send payload of packets switching DAP between JTAG and SWD mode.

Parameters

**swdctx* swd context pointer.

**ctlmsg* control message array pointer.

len number of elements to send from *ctlmsg.

Returns

number of elements appended (len), or SWD_ERROR_CODE on failure.

5.1.2.27 int swd_cmd_enqueue_mosi_dap_reset (swd_ctx_t * swdctx)

Append command queue with SW-DP-RESET sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.28 int swd_cmd_enqueue_mosi_data (swd_ctx_t * swdctx, int * data)

Append command queue with data and parity write.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.29 int swd_cmd_enqueue_mosi_data_ap (swd_ctx_t * swdctx, int * data)

Append command queue with data and automatic parity write.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.1.2.30 int swd_cmd_enqueue_mosi_data_p (swd_ctx_t * swdctx, int * data, char * parity)

Append command queue with data and provided parity write.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

**parity* parity value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.1.2.31 int swd_cmd_enqueue_mosi_idle (swd_ctx_t * swdctx)

Append command queue with idle sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.32 int swd_cmd_enqueue_mosi_jtag2swd (swd_ctx_t * swdctx)

Append command queue with JTAG-TO-SWD DAP-switch sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.33 int swd_cmd_enqueue_mosi_n_data_ap (swd_ctx_t * swdctx, int ** data, int count)

Append command queue with series of data and automatic parity writes.

Parameters

**swdctx* swd context pointer.

***data* data value array pointer.

count number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.1.2.34 int swd_cmd_enqueue_mosi_n_data_p (swd_ctx_t * swdctx, int ** data, char ** parity, int count)

Append command queue with series of data and provided parity writes.

Parameters

**swdctx* swd context pointer.

***data* data value array pointer.

***parity* parity value array pointer.

count number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.1.2.35 int swd_cmd_enqueue_mosi_nbit (swd_ctx_t * swdctx, char * data, int count)

Append command queue with bus binary write bit-by-bit operation.

This function will append command to the queue for each bit and store one bit into single char array element, so read is not constrained to 8 bits. On error memory is released and appropriate error code is returned. Important: Memory pointed by *data must be allocated prior call!

Parameters

**swdctx* swd context pointer.
***data* allocated data array to write result into.
count number of bits to read (also the ***data* size).

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.1.2.36 int swd_cmd_enqueue_mosi_parity (swd_ctx_t * swdctx, char * parity)

Append command queue with parity bit write.

Parameters

**swdctx* swd context pointer.
**parity* parity value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.37 int swd_cmd_enqueue_mosi_request (swd_ctx_t * swdctx, char * request)

Appends command queue with SWD Request packet header.

Note that contents is not validated, so bad request can be sent as well.

Parameters

**swdctx* swd context pointer.
**request* pointer to the 8-bit request payload.

Returns

return number elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.38 int swd_cmd_enqueue_mosi_swd2jtag (swd_ctx_t * swdctx)

Append command queue with SWD-TO-JTAG DAP-switch sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.1.2.39 int swd_cmd_enqueue_mosi_trn (swd_ctx_t * swdctx)

Append command queue with Turnaround activating MOSI mode.

Parameters

**swdctx* swd context pointer.

Returns

return number elements appended (1), or SWD_ERROR_CODE on failure.

5.1.2.40 int swd_cmdq_append (swd_cmd_t * cmdq, swd_cmd_t * cmd)

Append element pointed by *cmd at the end of the quque pointed by *cmdq.

After this operation queue will be pointed by appended element (ie. last element added becomes actual quque pointer to show what was added recently).

Parameters

**cmdq* pointer to any element on command queue

**cmd* pointer to the command to be appended

Returns

number of appended elements (one), SWD_ERROR_CODE on failure

5.1.2.41 swd_cmd_t* swd_cmdq_find_root (swd_cmd_t * cmdq)

Find queue root (first element).

Parameters

**cmdq* pointer to any queue element

Returns

swd_cmd_t* pointer to the first element (root), NULL on failure

5.1.2.42 swd_cmd_t* swd_cmdq_find_tail (swd_cmd_t * cmdq)

Find queue tail (last element).

Parameters

**cmdq* pointer to any queue element

Returns

swd_cmd_t* pointer to the last element (tail), NULL on failure

5.1.2.43 int swd_cmdq_flush (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

Flush command queue contents into interface driver.

Operation is specified by SWD_OPERATION and can be used to select how to flush the queue, ie. head-only, tail-only, one, all, etc.

Parameters

**swdctx* swd context pointer.

operation tells how to flush the queue.

Returns

number of commands transmitted, or SWD_ERROR_CODE on failure.

5.1.2.44 int swd_cmdq_free (swd_cmd_t * *cmdq*)

Free queue pointed by *cmdq element.

Parameters

**cmdq* pointer to any element on command queue

Returns

number of elements destroyed, SWD_ERROR_CODE on failure

5.1.2.45 int swd_cmdq_free_head (swd_cmd_t * *cmdq*)

Free queue head up to *cmdq element.

Parameters

**cmdq* pointer to the element that becomes new queue root.

Returns

number of elements destroyed, or SWD_ERROR_CODE on failure.

5.1.2.46 int swd_cmdq_free_tail (swd_cmd_t * *cmdq*)

Free queue tail starting after *cmdq element.

Parameters

**cmdq* pointer to the last element on the new queue.

Returns

number of elements destroyed, or SWD_ERROR_CODE on failure.

5.1.2.47 int swd_cmdq_init (swd_cmd_t * cmdq)

Initialize new queue element in memory that becomes a queue root.

Parameters

**cmdq* pointer to the command queue element of type [swd_cmd_t](#)

Returns

SWD_OK on success, SWD_ERROR_CODE code on failure

5.1.2.48 int swd_dap_detect (swd_ctx_t * swdctx, swd_operation_t operation, int ** idcode)

Macro: Reset target DAP, select SW-DP, read out IDCODE.

This is the proper SW-DP initialization as stated by ARM Information Center.

Parameters

**swdctx* swd context pointer.

operation type (SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE).

Returns

Target's IDCODE, or error code on failure.

5.1.2.49 int swd_dap_reset (swd_ctx_t * swdctx, swd_operation_t operation)

Debug Access Port Reset sends 50 CLK with TMS high that brings both SW-DP and JTAG-DP into reset state.

Parameters

**swdctx* swd context pointer.

operation type (SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE).

Returns

number of elements processed or SWD_ERROR_CODE code on failure.

5.1.2.50 int swd_dap_select (swd_ctx_t * swdctx, swd_operation_t operation)

Activate SW-DP by sending out RESET and JTAG-TO-SWD sequence on SWDIOTMS line.

Parameters

**swdctx* swd context.

Returns

number of control bytes executed, or error code on failre.

5.1.2.51 int swd_deinit (swd_ctx_t * *swdctx*)

De-initialize selected swd context and its command queue.

Parameters

**swdctx* swd context pointer.

Returns

number of elements freed, or SWD_ERROR_CODE on failure.

5.1.2.52 int swd_deinit_cmdq (swd_ctx_t * *swdctx*)

De-initialize command queue and free its memory on selected swd context.

Parameters

**swdctx* swd context pointer.

Returns

number of commands freed, or SWD_ERROR_CODE on failure.

5.1.2.53 int swd_deinit_ctx (swd_ctx_t * *swdctx*)

De-initialize selected swd context and free its memory.

Note: This function will not free command queue for selected context!

Parameters

**swdctx* swd context pointer.

Returns

SWD_OK on success, SWD_ERROR_CODE on failure.

5.1.2.54 int swd_dp_read_idcode (swd_ctx_t * *swdctx*, swd_operation_t *operation*, int ** *idcode*)

Macro: Read out IDCODE register and return its value on function return.

Parameters

**swdctx* swd context pointer.

operation operation type.

Returns

Target's IDCODE value or code error on failure.

5.1.2.55 `int swd_drv_transmit (swd_ctx_t * swdctx, swd_cmd_t * cmd)`

Transmit selected command from the command queue to the interface driver.

Parameters

- *swdctx* swd context pointer.
- *cmd* pointer to the command to be sent.

Returns

number of commands transmitted (1), or SWD_ERROR_CODE on failure.

5.1.2.56 `swd_ctx_t* swd_init (void)`

LibSWD initialization routine.

It should be called prior any operation made with libswd. It initializes command queue and basic parameters for context that is returned as pointer.

Returns

pointer to the initialized swd context.

5.1.2.57 `int swd_log (swd_ctx_t * swdctx, swd_loglevel_t loglevel, char * msg, ...)`

Put a message into swd context log at specified verbosity level.

If specified message's log level is lower than actual context configuration, message will be omitted. Verbosity level increases from 0 (silent) to 4 (debug).

Parameters

- *swdctx* swd context.
- loglevel* at which to put selected message.
- *msg* message body with variable arguments as in "printf".

Returns

number of characters written or error code on failure.

5.1.2.58 `int swd_log_level_set (swd_ctx_t * swdctx, swd_loglevel_t loglevel)`

Change log level to increase or decrease verbosity level.

Parameters

- *swdctx* swd context.
- loglevel* is the target verbosity level to be set.

Returns

SWD_OK on success or error code.

5.2 libswd.h File Reference

Classes

- struct [swd_cmd_t](#)
SWD Command Element Structure.
- struct [swd_context_config_t](#)
Context configuration structure.
- struct [swd_swdp_t](#)
Most actual Serial Wire Debug Port Registers.
- struct [swd_ahbap_t](#)
Most actual Advanced High Bandwidth Access Peripheral Bus Registers.
- struct [swd_transaction_t](#)
Most actual SWD bus transaction/packet data.
- struct [swd_driver_t](#)
Interface Driver structure.
- struct [swd_ctx_t](#)
SWD Context Structure definition.

Defines

- #define [SWD_REQUEST_START_BITNUM](#) 7
SWD Packets Bit Fields and Values.
- #define [SWD_REQUEST_APnDP_BITNUM](#) 6
Access Port (high) or Debug Port (low) access.
- #define [SWD_REQUEST_RnW_BITNUM](#) 5
Read (high) or Write (low) access.
- #define [SWD_REQUEST_ADDR_BITNUM](#) 4
LSB of the address field in request header.
- #define [SWD_REQUEST_A2_BITNUM](#) 4
Target Register Address bit 2.
- #define [SWD_REQUEST_A3_BITNUM](#) 3
Target Register Address bit 3.
- #define [SWD_REQUEST_PARITY_BITNUM](#) 2
Odd Parity calculated from APnDP, RnW, A[2:3].

- #define [SWD_REQUEST_STOP_BITNUM](#) 1
Packet Stop bit, always 0.
- #define [SWD_REQUEST_PARK_BITNUM](#) 0
Park wire and switch between receive/transmit.
- #define [SWD_REQUEST_START_VAL](#) 1
Start Bit Value is always 1.
- #define [SWD_REQUEST_STOP_VAL](#) 0
Stop Bit Value is always 0.
- #define [SWD_REQUEST_PARK_VAL](#) 1
Park bus and put outputs into Hi-Z state.
- #define [SWD_REQUEST_BITLEN](#) 8
Number of bits in request packet header.
- #define [SWD_ADDR_MINVAL](#) 0
Address field minimal value.
- #define [SWD_ADDR_MAXVAL](#) 3
Address field maximal value.
- #define [SWD_ACK_BITLEN](#) 3
Number of bits in Acknowledge packet.
- #define [SWD_ACK_OK_VAL](#) 4
OK code value.
- #define [SWD_ACK_WAIT_VAL](#) 2
WAIT code value.
- #define [SWD_ACK_FAULT_VAL](#) 1
FAULT code value.
- #define [SWD_DP_ADDR_IDCODE](#) 0
IDCODE register address (RO).
- #define [SWD_DP_ADDR_ABORT](#) 0
ABORT register address (WO).
- #define [SWD_DP_ADDR_CTRLSTAT](#) 1
CTRLSTAT register address (R/W, CTRLSEL=b0).
- #define [SWD_DP_ADDR_WCR](#) 1
WCR register address (R/W, CTRLSEL=b1).
- #define [SWD_DP_ADDR_RESEND](#) 2
RESEND register address (RO).

- #define [SWD_DP_ADDR_SELECT](#) 2
SELECT register address (WO).
- #define [SWD_DP_ADDR_RDBUF](#) 3
RDBUF register address (RO).
- #define [SWD_ABORT_BITNUM_DAPABORT](#) 0
SW-DP ABORT Register map.
- #define [SWD_ABORT_BITNUM_DSTKCMPLR](#) 1
DSTKCMPLR bit number.
- #define [SWD_ABORT_BITNUM_DSTKERRCLR](#) 2
DSTKERRCLR bit number.
- #define [SWD_ABORT_BITNUM_DWDERRCLR](#) 3
DWDERRCLR bit number.
- #define [SWD_ABORT_BITNUM_DORUNERRCLR](#) 4
DORUNERRCLR bit number.
- #define [SWD_CTRLSTAT_BITNUM_ORUNDETECT](#) 0
SW-DP CTRL/STAT Register map.
- #define [SWD_CTRLSTAT_BITNUM_OSTICKYORUN](#) 1
OSTICKYORUN bit number.
- #define [SWD_CTRLSTAT_BITNUM_OTRNMODE](#) 2
OTRNMODE bit number.
- #define [SWD_CTRLSTAT_BITNUM_OSTICKYCMP](#) 4
OSTICKYCMP bit number.
- #define [SWD_CTRLSTAT_BITNUM_OSTICKYERR](#) 5
OSTICKYERR bit number.
- #define [SWD_CTRLSTAT_BITNUM_OREADOK](#) 6
OREADOK bit number.
- #define [SWD_CTRLSTAT_BITNUM_OWDATAERR](#) 7
OWDATAERR bit number.
- #define [SWD_CTRLSTAT_BITNUM_OMASKLANE](#) 8
OMASKLANE bit number.
- #define [SWD_CTRLSTAT_BITNUM_OTRNCNT](#) 12
OTRNCNT bit number.
- #define [SWD_CTRLSTAT_BITNUM_OCDBGRESTREQ](#) 26

OCDBGIRSTREQ bit number.

- #define [SWD_CTRLSTAT_BITNUM_OCDBGIRSTACK](#) 27
OCDBGIRSTACK bit number.
- #define [SWD_CTRLSTAT_BITNUM_OCDBGPWRUPREQ](#) 28
OCDBGPWRUPREQ bit number.
- #define [SWD_CTRLSTAT_BITNUM_OCDBGPWRUPACK](#) 29
OCDBGPWRUPACK bit number.
- #define [SWD_CTRLSTAT_BITNUM_OCSYSPWRUPREQ](#) 30
OCSYSPWRUPREQ bit number.
- #define [SWD_CTRLSTAT_BITNUM_OCSYSPWRUPACK](#) 31
OCSYSPWRUPACK bit number.
- #define [SWD_MASKLANE_0](#) 0b0001
SW-DP CTRLSTAT MASKLANE available values.
- #define [SWD_MASKLANE_1](#) 0b0010
Compare byte lane 1 (0x---FF--).
- #define [SWD_MASKLANE_2](#) 0b0100
Compare byte lane 2 (0x--FF----).
- #define [SWD_MASKLANE_3](#) 0b1000
Compare byte lane 3 (0xFF-----).
- #define [SWD_SELECT_BITNUM_CTRLSEL](#) 0
SW-DP SELECT Register map.
- #define [SWD_SELECT_BITNUM_APBANKSEL](#) 4
APBANKSEL bit number.
- #define [SWD_SELECT_BITNUM_APSEL](#) 24
APSEL bit number.
- #define [SWD_WCR_BITNUM_PRESCALER](#) 0
SW-DP WCR Register map.
- #define [SWD_WCR_BITNUM_WIREMODE](#) 6
- #define [SWD_WCR_BITNUM_TURNROUND](#) 8
- #define [SWD_TURNROUND_1_CODE](#) 0
SW-DP WCR TURNROUND available values.
- #define **SWD_TURNROUND_1_VAL** 1
- #define [SWD_TURNROUND_2_CODE](#) 1
- #define **SWD_TURNROUNT_2_VAL** 2
- #define [SWD_TURNROUND_3_CODE](#) 2

- #define **SWD_TURNROUND_3_VAL** 3
- #define **SWD_TURNROUND_4_CODE** 3
- #define **SWD_TURNROUND_4_VAL** 4
- #define **SWD_TURNROUND_MIN_VAL** SWD_TURNROUND_1_VAL
- #define **SWD_TURNROUND_MIN_CODE** SWD_TURNROUND_1_CODE
- #define **SWD_TURNROUND_MAX_VAL** SWD_TURNROUND_4_VAL
- #define **SWD_TURNROUND_MAX_CODE** SWD_TURNROUND_4_CODE
- #define **SWD_TURNROUND_DEFAULT_VAL** SWD_TURNROUND_1_VAL
- #define **AHB_AP_CONTROLSTATUS** 0x00

AHB-AP Registers Map.

- #define **AHB_AP_TAR** 0x04
R/W, 32bit, reset value: 0x00000000.
- #define **AHB_AP_DRW** 0x0C
R/W, 32bit.
- #define **AHB_AP_BD0** 0x10
R/W, 32bit.
- #define **AHB_AP_BD1** 0x14
R/W, 32bit.
- #define **AHB_AP_BD2** 0x18
R/W, 32bit.
- #define **AHB_AP_BD3** 0x1C
R/W, 32bit.
- #define **AHB_AP_DROMT** 0xF8
RO, 32bit, reset value: 0xE0FF000.
- #define **AHB_AP_IDR** 0xFC
RO, 32bit, reset value: 0x24770001.
- #define **SWD_DATA_MAXBITCOUNT** 32
SWD queue and payload data definitions.
- #define **SWD_DATA_BYTESIZE** 8
How many bits are there in a byte.
- #define **SWD_DATA_BITLEN** 32
How many bits are there in data payload.
- #define **SWD_CMDQLEN_DEFAULT** 1024;
How long is the command queue by default.

Typedefs

- typedef struct [swd_cmd_t](#) [swd_cmd_t](#)
SWD Command Element Structure.

Enumerations

- enum [swd_error_code_t](#) {
[SWD_OK](#) = 0, [SWD_ERROR_GENERAL](#) = -1, [SWD_ERROR_NULLPOINTER](#) = -2, [SWD_ERROR_NULLQUEUE](#) = -3,
[SWD_ERROR_NULLTRN](#) = -4, [SWD_ERROR_PARAM](#) = -5, [SWD_ERROR_OUTOFMEM](#) = -6,
[SWD_ERROR_RESULT](#) = -7,
[SWD_ERROR_RANGE](#) = -8, [SWD_ERROR_DEFINITION](#) = -9, [SWD_ERROR_NULLCONTEXT](#) = -10, [SWD_ERROR_QUEUE](#) = -11,
[SWD_ERROR_ADDR](#) = -12, [SWD_ERROR_APnDP](#) = -13, [SWD_ERROR_RnW](#) = -14, [SWD_ERROR_PARITY](#) = -15,
[SWD_ERROR_ACK](#) = -16, [SWD_ERROR_ACKUNKNOWN](#) = -19, [SWD_ERROR_ACKNOTDONE](#) = -20, [SWD_ERROR_ACKMISSING](#) = -21,
[SWD_ERROR_ACKMISMATCH](#) = -22, [SWD_ERROR_ACKORDER](#) = -23, [SWD_ERROR_BADOPCODE](#) = -24, [SWD_ERROR_NODATACMD](#) = -25,
[SWD_ERROR_DATAPTR](#) = -26, [SWD_ERROR_NOPARITYCMD](#) = -27, [SWD_ERROR_PARITYPTR](#) = -28, [SWD_ERROR_NOTDONE](#) = -29,
[SWD_ERROR_QUEUEEROOT](#) = -30, [SWD_ERROR_QUEUETAIL](#) = -31, [SWD_ERROR_BADCMDTYPE](#) = -32, [SWD_ERROR_BADCMDDATA](#) = -33,
[SWD_ERROR_TURNAROUND](#) = -34, [SWD_ERROR_DRIVER](#) = -35, [SWD_ERROR_ACK_WAIT](#) = -36, [SWD_ERROR_ACK_FAULT](#) = -37,
[SWD_ERROR_QUEUENOTFREE](#) = -38, [SWD_ERROR_TRANSPORT](#) = -39, [SWD_ERROR_DIRECTION](#) = -40, [SWD_ERROR_LOGLEVEL](#) = -41 }
Status and Error Codes definitions.

- enum [swd_loglevel_t](#) {
[SWD_LOGLEVEL_MIN](#) = 0, [SWD_LOGLEVEL_SILENT](#) = 0, [SWD_LOGLEVEL_ERROR](#) = 1, [SWD_LOGLEVEL_WARNING](#) = 2,
[SWD_LOGLEVEL_NORMAL](#) = 3, [SWD_LOGLEVEL_INFO](#) = 4, [SWD_LOGLEVEL_DEBUG](#) = 5, [SWD_LOGLEVEL_MAX](#) = 5 }
Logging Level Codes definition.

- enum [swd_cmdtype_t](#) {
[SWD_CMDTYPE_MOSI_DATA](#) = -7, [SWD_CMDTYPE_MOSI_REQUEST](#) = -6, [SWD_CMDTYPE_MOSI_TRN](#) = -5, [SWD_CMDTYPE_MOSI_PARITY](#) = -4,
[SWD_CMDTYPE_MOSI_BITBANG](#) = -3, [SWD_CMDTYPE_MOSI_CONTROL](#) = -2, [SWD_CMDTYPE_MOSI](#) = -1, [SWD_CMDTYPE_UNDEFINED](#) = 0,
[SWD_CMDTYPE_MISO](#) = 1, [SWD_CMDTYPE_MISO_ACK](#) = 2, [SWD_CMDTYPE_MISO_BITBANG](#) = 3, [SWD_CMDTYPE_MISO_PARITY](#) = 4,
[SWD_CMDTYPE_MISO_TRN](#) = 5, [SWD_CMDTYPE_MISO_DATA](#) = 6 }
SWD Command Codes definitions.

- enum `swd_shiftdir_t` { `SWD_DIR_LSBFIRST` = 0, `SWD_DIR_MSBFIRST` = 1 }
What is the shift direction LSB-first or MSB-first.
- enum `swd_operation_t` {
 `SWD_OPERATION_FIRST` = 1, `SWD_OPERATION_ENQUEUE` = 1, `SWD_OPERATION_EXECUTE` = 2, `SWD_OPERATION_TRANSMIT_HEAD` = 3,
 `SWD_OPERATION_TRANSMIT_TAIL` = 4, `SWD_OPERATION_TRANSMIT_ALL` = 5, `SWD_OPERATION_TRANSMIT_ONE` = 6, `SWD_OPERATION_TRANSMIT_LAST` = 7,
 `SWD_OPERATION_LAST` = 7 }
Command Queue operations codes.
- enum `swd_bool_t` { `SWD_FALSE` = 0, `SWD_TRUE` = 1 }
Boolean values definition.

Functions

- int `swd_bin8_parity_even` (char *data, char *parity)
Some comments on the function behavior.
- int `swd_bin32_parity_even` (int *data, char *parity)
Data parity calculator, calculates even parity on integer type.
- int `swd_bin8_print` (char *data)
Prints binary data of a char value on the screen.
- int `swd_bin32_print` (int *data)
Prints binary data of an integer value on the screen.
- char * `swd_bin8_string` (char *data)
Generates string containing binary data of a char value.
- char * `swd_bin32_string` (int *data)
Generates string containing binary data of an integer value.
- int `swd_bin8_bitswap` (unsigned char *buffer, int bitcount)
*Bit swap helper function that reverse bit order in char *buffer.*
- int `swd_bin32_bitswap` (unsigned int *buffer, int bitcount)
*Bit swap helper function that reverse bit order in int *buffer.*
- int `swd_cmdq_init` (swd_cmd_t *cmdq)
Initialize new queue element in memory that becomes a queue root.
- swd_cmd_t * `swd_cmdq_find_root` (swd_cmd_t *cmdq)
Find queue root (first element).
- swd_cmd_t * `swd_cmdq_find_tail` (swd_cmd_t *cmdq)

Find queue tail (last element).

- `int swd_cmdq_append (swd_cmd_t *cmdq, swd_cmd_t *cmd)`
*Append element pointed by *cmd at the end of the quque pointed by *cmdq.*
- `int swd_cmdq_free (swd_cmd_t *cmdq)`
*Free queue pointed by *cmdq element.*
- `int swd_cmdq_free_head (swd_cmd_t *cmdq)`
*Free queue head up to *cmdq element.*
- `int swd_cmdq_free_tail (swd_cmd_t *cmdq)`
*Free queue tail starting after *cmdq element.*
- `int swd_cmdq_flush (swd_ctx_t *swdctx, swd_operation_t operation)`
Flush command queue contents into interface driver.
- `int swd_cmd_enqueue (swd_ctx_t *swdctx, swd_cmd_t *cmd)`
Append selected command to a context's command queue.
- `int swd_cmd_enqueue_mosi_request (swd_ctx_t *swdctx, char *request)`
Appends command queue with SWD Request packet header.
- `int swd_cmd_enqueue_mosi_trn (swd_ctx_t *swdctx)`
Append command queue with Turnaround activating MOSI mode.
- `int swd_cmd_enqueue_miso_trn (swd_ctx_t *swdctx)`
Append command queue with Turnaround activating MISO mode.
- `int swd_cmd_enqueue_miso_nbit (swd_ctx_t *swdctx, char **data, int count)`
Append command queue with bus binary read bit-by-bit operation.
- `int swd_cmd_enqueue_mosi_nbit (swd_ctx_t *swdctx, char *data, int count)`
Append command queue with bus binary write bit-by-bit operation.
- `int swd_cmd_enqueue_mosi_parity (swd_ctx_t *swdctx, char *parity)`
Append command queue with parity bit write.
- `int swd_cmd_enqueue_miso_parity (swd_ctx_t *swdctx, char **parity)`
Append command queue with parity bit read.
- `int swd_cmd_enqueue_miso_data (swd_ctx_t *swdctx, int **data)`
Append command queue with data read.
- `int swd_cmd_enqueue_miso_data_p (swd_ctx_t *swdctx, int **data, char **parity)`
Append command queue with data and parity read.
- `int swd_cmd_enqueue_miso_n_data_p (swd_ctx_t *swdctx, int **data, char **parity, int count)`
Append command queue with series of data and parity read.

- int [swd_cmd_enqueue_mosi_data](#) (swd_ctx_t *swdctx, int *data)
Append command queue with data and parity write.
- int [swd_cmd_enqueue_mosi_data_ap](#) (swd_ctx_t *swdctx, int *data)
Append command queue with data and automatic parity write.
- int [swd_cmd_enqueue_mosi_data_p](#) (swd_ctx_t *swdctx, int *data, char *parity)
Append command queue with data and provided parity write.
- int [swd_cmd_enqueue_mosi_n_data_ap](#) (swd_ctx_t *swdctx, int **data, int count)
Append command queue with series of data and automatic parity writes.
- int [swd_cmd_enqueue_mosi_n_data_p](#) (swd_ctx_t *swdctx, int **data, char **parity, int count)
Append command queue with series of data and provided parity writes.
- int [swd_cmd_enqueue_miso_ack](#) (swd_ctx_t *swdctx, char **ack)
Append queue with ACK read.
- int [swd_cmd_enqueue_mosi_control](#) (swd_ctx_t *swdctx, char *ctlmsg, int len)
Append command queue with len-octet size control sequence.
- int [swd_cmd_enqueue_mosi_dap_reset](#) (swd_ctx_t *swdctx)
Append command queue with SW-DP-RESET sequence.
- int [swd_cmd_enqueue_mosi_idle](#) (swd_ctx_t *swdctx)
Append command queue with idle sequence.
- int [swd_cmd_enqueue_mosi_jtag2swd](#) (swd_ctx_t *swdctx)
Append command queue with JTAG-TO-SWD DAP-switch sequence.
- int [swd_cmd_enqueue_mosi_swd2jtag](#) (swd_ctx_t *swdctx)
Append command queue with SWD-TO-JTAG DAP-switch sequence.
- char * [swd_cmd_string_cmdtype](#) (swd_cmd_t *cmd)
- int [swd_bus_setdir_mosi](#) (swd_ctx_t *swdctx)
Append command queue with TRN WRITE/MOSI, if previous command was READ/MISO.
- int [swd_bus_setdir_miso](#) (swd_ctx_t *swdctx)
Append command queue with TRN READ/MISO, if previous command was WRITE/MOSI.
- int [swd_bus_write_request](#) (swd_ctx_t *swdctx, [swd_operation_t](#) operation, char *APnDP, char *RnW, char *addr)
Perform Request.
- int [swd_bus_read_ack](#) (swd_ctx_t *swdctx, [swd_operation_t](#) operation, char **ack)
*Perform ACK read into *ack and verify received data.*
- int [swd_bus_write_data_p](#) (swd_ctx_t *swdctx, [swd_operation_t](#) operation, int *data, char *parity)
Perform (MOSI) data write with provided parity value.

- int [swd_bus_write_data_ap](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation, int *data)
Perform (MOSI) data write with automatic parity calculation.
- int [swd_bus_read_data_p](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation, int **data, char **parity)
Perform (MISO) data read.
- int [swd_bus_write_control](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation, char *ctlmsg, int len)
Write CONTROL byte to the Target's DAP.
- int [swd_bitgen8_request](#) ([swd_ctx_t](#) *swdctx, char *APnDP, char *RnW, char *addr, char *request)
Generate 8-bit SWD-REQUEST packet contents with provided parameters.
- int [swd_drv_transmit](#) ([swd_ctx_t](#) *swdctx, [swd_cmd_t](#) *cmd)
Transmit selected command from the command queue to the interface driver.
- int [swd_drv_mosi_8](#) ([swd_ctx_t](#) *swdctx, [swd_cmd_t](#) *cmd, char *data, int bits, int nLSBfirst)
- int [swd_drv_mosi_32](#) ([swd_ctx_t](#) *swdctx, [swd_cmd_t](#) *cmd, int *data, int bits, int nLSBfirst)
- int [swd_drv_miso_8](#) ([swd_ctx_t](#) *swdctx, [swd_cmd_t](#) *cmd, char *data, int bits, int nLSBfirst)
- int [swd_drv_miso_32](#) ([swd_ctx_t](#) *swdctx, [swd_cmd_t](#) *cmd, int *data, int bits, int nLSBfirst)
- int [swd_drv_mosi_trn](#) ([swd_ctx_t](#) *swdctx, int clks)
- int [swd_drv_miso_trn](#) ([swd_ctx_t](#) *swdctx, int clks)
- int [swd_dap_reset](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation)
Debug Access Port Reset sends 50 CLK with TMS high that brings both SW-DP and JTAG-DP into reset state.
- int [swd_dap_select](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation)
Activate SW-DP by sending out RESET and JTAG-TO-SWD sequence on SWDIOTMS line.
- int [swd_dap_detect](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation, int **idcode)
Macro: Reset target DAP, select SW-DP, read out IDCODE.
- int [swd_dp_read_idcode](#) ([swd_ctx_t](#) *swdctx, [swd_operation_t](#) operation, int **idcode)
Macro: Read out IDCODE register and return its value on function return.
- int [swd_log](#) ([swd_ctx_t](#) *swdctx, [swd_loglevel_t](#) loglevel, char *msg,...)
Put a message into swd context log at specified verbosity level.
- int [swd_log_level_set](#) ([swd_ctx_t](#) *swdctx, [swd_loglevel_t](#) loglevel)
Change log level to increase or decrease verbosity level.
- int [swd_log_level_inherit](#) ([swd_ctx_t](#) *swdctx, int loglevel)
- char * [swd_error_string](#) ([swd_error_code_t](#) error)
- [swd_ctx_t](#) * [swd_init](#) (void)
LibSWD initialization routine.
- int [swd_deinit_ctx](#) ([swd_ctx_t](#) *swdctx)
De-initialize selected swd context and free its memory.

- int [swd_deinit_cmdq](#) (swd_ctx_t *swdctx)
De-initialize command queue and free its memory on selected swd context.
- int [swd_deinit](#) (swd_ctx_t *swdctx)
De-initialize selected swd context and its command queue.

5.2.1 Detailed Description

5.2.2 Define Documentation

5.2.2.1 #define AHB_AP_BD0 0x10

R/W, 32bit.

R/W, 32bit

5.2.2.2 #define AHB_AP_BD1 0x14

R/W, 32bit.

R/W, 32bit

5.2.2.3 #define AHB_AP_BD2 0x18

R/W, 32bit.

R/W, 32bit

5.2.2.4 #define AHB_AP_BD3 0x1C

R/W, 32bit.

R/W, 32bit

5.2.2.5 #define AHB_AP_CONTROLSTATUS 0x00

AHB-AP Registers Map.

TODO!!!! R/W, 32bit, reset value: 0x43800042 R/W, 32bit, reset value: 0x43800042

5.2.2.6 #define AHB_AP_DROMT 0xF8

RO, 32bit, reset value: 0xE00FF000.

RO, 32bit, reset value: 0xE00FF000

5.2.2.7 #define AHB_AP_DRW 0x0C

R/W, 32bit.

R/W, 32bit

5.2.2.8 #define AHB_AP_IDR 0xFC

RO, 32bit, reset value: 0x24770001.

RO, 32bit, reset value: 0x24770001

5.2.2.9 #define AHB_AP_TAR 0x04

R/W, 32bit, reset value: 0x00000000.

R/W, 32bit, reset value: 0x00000000

5.2.2.10 #define SWD_ABORT_BITNUM_DAPABORT 0

SW-DP ABORT Register map.

DAPABORT bit number.

5.2.2.11 #define SWD_CTRLSTAT_BITNUM_ORUNDETECT 0

SW-DP CTRL/STAT Register map.

ORUNDETECT bit number.

5.2.2.12 #define SWD_DATA_MAXBITCOUNT 32

SWD queue and payload data definitions.

What is the maximal bit length of the data.

5.2.2.13 #define SWD_MASKLANE_0 0b0001

SW-DP CTRLSTAT MASKLANE available values.

Compare byte lane 0 (0x-----FF)

5.2.2.14 #define SWD_REQUEST_START_BITNUM 7

SWD Packets Bit Fields and Values.

Packet Start bit, always set to 1.

5.2.2.15 #define SWD_SELECT_BITNUM_CTRLSEL 0

SW-DP SELECT Register map.

CTRLSEL bit number.

5.2.2.16 #define SWD_TURNROUND_1_CODE 0

SW-DP WCR TURNROUND available values.

TRN takes one CLK cycle. TRN takes one CLK cycle.

5.2.2.17 #define SWD_TURNROUND_2_CODE 1

TRN takes two CLK cycles.

5.2.2.18 #define SWD_TURNROUND_3_CODE 2

TRN takes three CLK cycles.

5.2.2.19 #define SWD_TURNROUND_4_CODE 3

TRN takes four CLK cycles. ????

5.2.2.20 #define SWD_TURNROUND_DEFAULT_VAL SWD_TURNROUND_1_VAL

Default TRN length is one CLK.

5.2.2.21 #define SWD_TURNROUND_MAX_VAL SWD_TURNROUND_4_VAL

longest TRN time.

5.2.2.22 #define SWD_TURNROUND_MIN_VAL SWD_TURNROUND_1_VAL

shortest TRN time.

5.2.2.23 #define SWD_WCR_BITNUM_PRESCALER 0

SW-DP WCR Register map.

PRESCALER bit number. PRESCALER bit number.

5.2.2.24 #define SWD_WCR_BITNUM_TURNROUND 8

TURNROUND bit number.

5.2.2.25 #define SWD_WCR_BITNUM_WIREMODE 6

WIREMODE bit number.

5.2.3 Typedef Documentation**5.2.3.1 typedef struct swd_cmd_t swd_cmd_t**

SWD Command Element Structure.

In libswd each operation is split into separate commands (request, trn, ack, data, parity) that can be appended to the command queue and later executed. This organization allows better granularity for tracing bugs and makes possible to compose complete bus/target operations made of simple commands.

5.2.4 Enumeration Type Documentation

5.2.4.1 enum swd_bool_t

Boolean values definition.

Enumerator:

SWD_FALSE False is 0.

SWD_TRUE True is 1.

5.2.4.2 enum swd_cmdtype_t

SWD Command Codes definitions.

Available values: MISO>0, MOSI<0, undefined=0. To check command direction (read/write) multiply tested value with one of the MOSI or MISO commands

- result is positive for equal direction and negative if direction differs. Command Type codes definition, use this to see names in debugger.

Enumerator:

SWD_CMDTYPE_MOSI_DATA Contains MOSI data (from host).

SWD_CMDTYPE_MOSI_REQUEST Contains MOSI request packet.

SWD_CMDTYPE_MOSI_TRN Bus will switch into MOSI mode.

SWD_CMDTYPE_MOSI_PARITY Contains MOSI data parity.

SWD_CMDTYPE_MOSI_BITBANG Allows MOSI operation bit-by-bit.

SWD_CMDTYPE_MOSI_CONTROL MOSI control sequence (ie. sw-dp reset, idle).

SWD_CMDTYPE_MOSI Master Output Slave Input direction.

SWD_CMDTYPE_UNDEFINED undefined command, not transmitted.

SWD_CMDTYPE_MISO Master Input Slave Output direction.

SWD_CMDTYPE_MISO_ACK Contains ACK data from target.

SWD_CMDTYPE_MISO_BITBANG Allows MISO operation bit-by-bit.

SWD_CMDTYPE_MISO_PARITY Contains MISO data parity.

SWD_CMDTYPE_MISO_TRN Bus will switch into MISO mode.

SWD_CMDTYPE_MISO_DATA Contains MISO data (from target).

5.2.4.3 enum swd_error_code_t

Status and Error Codes definitions.

Error Codes definition, use this to have its name on debugger.

Enumerator:

SWD_OK No error.

SWD_ERROR_GENERAL General error.

SWD_ERROR_NULLPOINTER Null pointer.

SWD_ERROR_NULLQUEUE Null queue pointer.

SWD_ERROR_NULLTRN Null TurnaRouNd pointer.

SWD_ERROR_PARAM Bad parameter.

SWD_ERROR_OUTOFMEM Out of memory.

SWD_ERROR_RESULT Bad result.

SWD_ERROR_RANGE Out of range.

SWD_ERROR_DEFINITION Definition (internal) error.

SWD_ERROR_NULLCONTEXT Null context pointer.

SWD_ERROR_QUEUE Queue error.

SWD_ERROR_ADDR Addressing error.

SWD_ERROR_APnDP Bad APnDP value.

SWD_ERROR_RnW Bad RnW value.

SWD_ERROR_PARITY Parity error.

SWD_ERROR_ACK Acknowledge error.

SWD_ERROR_ACKUNKNOWN Unknown ACK value.

SWD_ERROR_ACKNOTDONE ACK not yet executed on target.

SWD_ERROR_ACKMISSING ACK command not found on the queue.

SWD_ERROR_ACKMISMATCH Bad ACK result address.

SWD_ERROR_ACKORDER ACK not in order REQ->TRN->ACK.

SWD_ERROR_BADOPCODE Unsupported operation requested.

SWD_ERROR_NODATACMD Command not found on the queue.

SWD_ERROR_DATAPTR Bad DATA pointer address.

SWD_ERROR_NOPARITYCMD Parity after Data missing or misplaced.

SWD_ERROR_PARITYPTR Bad PARITY pointer address.

SWD_ERROR_NOTDONE Could not end selected task.

SWD_ERROR_QUEUEROOT Queue root not found or null.

SWD_ERROR_QUEUETAIL Queue tail not found or null.

SWD_ERROR_BADCMDTYPE Unknown command detected.

SWD_ERROR_BADCMDDATA Bad command data.

SWD_ERROR_TURNAROUND Error during turnaround switch.

SWD_ERROR_DRIVER Driver error.

SWD_ERROR_ACK_WAIT Received ACK WAIT.

SWD_ERROR_ACK_FAULT Received ACK FAULT.

SWD_ERROR_QUEUENOTFREE Cannot free resources, queue not empty.

SWD_ERROR_TRANSPORT Transport type unknown or undefined.

SWD_ERROR_DIRECTION Direction error (LSb/MSb first).

SWD_ERROR_LOGLEVEL Invalid loglevel number.

5.2.4.4 enum swd_loglevel_t

Logging Level Codes definition.

Logging Level codes definition, use this to have its name on debugger.

Enumerator:

SWD_LOGLEVEL_SILENT Remain silent.
SWD_LOGLEVEL_ERROR Show errors.
SWD_LOGLEVEL_WARNING Show warnings.
SWD_LOGLEVEL_NORMAL Normal verbosity.
SWD_LOGLEVEL_INFO Show messages.
SWD_LOGLEVEL_DEBUG Show all including debug information.

5.2.4.5 enum swd_operation_t

Command Queue operations codes.

Enumerator:

SWD_OPERATION_FIRST First operation to know its code.
SWD_OPERATION_ENQUEUE Append command(s) to the queue.
SWD_OPERATION_EXECUTE Queue commands then flush the queue.
SWD_OPERATION_TRANSMIT_HEAD Transmit root..current (head).
SWD_OPERATION_TRANSMIT_TAIL Transmit current..last (tail).
SWD_OPERATION_TRANSMIT_ALL Transmit all commands on the queue.
SWD_OPERATION_TRANSMIT_ONE Transmit only current command.
SWD_OPERATION_TRANSMIT_LAST Transmit last command on the queue.
SWD_OPERATION_LAST Last operation to know its code.

5.2.4.6 enum swd_shiftdir_t

What is the shift direction LSB-first or MSB-first.

Enumerator:

SWD_DIR_LSBFIRST Data is shifted in/out right (LSB-first).
SWD_DIR_MSBFIRST Data is shifted in/out left (MSB-first).

5.2.5 Function Documentation

5.2.5.1 int swd_bin32_bitswap (unsigned int * *buffer*, int *bitcount*)

Bit swap helper function that reverse bit order in int *buffer.

Most Significant Bit becomes Least Significant Bit. It is possible to swap only n-bits from int (32-bit) *buffer.

Parameters

**buffer* unsigned char (32-bit) data pointer.

bitcount how many bits to swap.

Returns

swapped bit count (positive) or error code (negative).

5.2.5.2 int swd_bin32_parity_even (int * data, char * parity)

Data parity calculator, calculates even parity on integer type.

Parameters

**data* source data pointer.

**parity* resulting data pointer.

Returns

negative value on error, 0 or 1 as parity result.

5.2.5.3 int swd_bin32_print (int * data)

Prints binary data of an integer value on the screen.

Parameters

**data* source data pointer.

Returns

number of characters printed.

5.2.5.4 char* swd_bin32_string (int * data)

Generates string containing binary data of an integer value.

Parameters

**data* source data pointer.

Returns

pointer to the resulting string.

5.2.5.5 int swd_bin8_bitswap (unsigned char * *buffer*, int *bitcount*)

Bit swap helper function that reverse bit order in char *buffer.

Most Significant Bit becomes Least Significant Bit. It is possible to swap only n-bits from char (8-bit) *buffer.

Parameters

**buffer* unsigned char (8-bit) data pointer.

bitcount how many bits to swap.

Returns

swapped bit count (positive) or error code (negative).

5.2.5.6 int swd_bin8_parity_even (char * *data*, char * *parity*)

Some comments on the function behavior.

Some comments on the function behavior.

Parameters

**data* source data pointer.

**parity* resulting data pointer.

Returns

negative value on error, 0 or 1 as parity result.

5.2.5.7 int swd_bin8_print (char * *data*)

Prints binary data of a char value on the screen.

Parameters

**data* source data pointer.

Returns

number of characters printed.

5.2.5.8 char* swd_bin8_string (char * *data*)

Generates string containing binary data of a char value.

Parameters

**data* source data pointer.

Returns

pointer to the resulting string.

5.2.5.9 `int swd_bitgen8_request (swd_ctx_t * swdctx, char * APnDP, char * RnW, char * addr, char * request)`

Generate 8-bit SWD-REQUEST packet contents with provided parameters.

Note that parity bit value is calculated automatically.

Parameters

- *swdctx* swd context pointer.
- *APnDP* AccessPort (high) or DebugPort (low) access type pointer.
- *RnW* Read (high) or Write (low) operation type pointer.
- *addr* target register address value pointer.
- *request* pointer where to store resulting packet.

Returns

number of generated packets (1), or SWD_ERROR_CODE on failure.

5.2.5.10 `int swd_bus_read_ack (swd_ctx_t * swdctx, swd_operation_t operation, char ** ack)`

Perform ACK read into *ack and verify received data.

Parameters

- *swdctx* swd context pointer.
- operation* type of action to perform with generated request.
- *ack* pointer to the result location.

Returns

number of commands processed, or SWD_ERROR_CODE on failure.

5.2.5.11 `int swd_bus_read_data_p (swd_ctx_t * swdctx, swd_operation_t operation, int ** data, char ** parity)`

Perform (MISO) data read.

Parameters

- *swdctx* swd context pointer.
- operation* type of action to perform on generated command.
- *data* payload value pointer.
- *parity* payload parity value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.2.5.12 int swd_bus_setdir_miso (swd_ctx_t * swdctx)

Append command queue with TRN READ/MISO, if previous command was WRITE/MOSI.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.13 int swd_bus_setdir_mosi (swd_ctx_t * swdctx)

Append command queue with TRN WRITE/MOSI, if previous command was READ/MISO.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.14 int swd_bus_write_control (swd_ctx_t * swdctx, swd_operation_t operation, char * ctlmsg, int len)

Write CONTROL byte to the Target's DAP.

Parameters

**swdctx* swd context.

operation can be SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE.

**ctlmsg* byte/char array that contains control payload.

len number of bytes in the **ctlmsg* to send.

Returns

number of bytes sent or SWD_ERROR_CODE on failure.

5.2.5.15 int swd_bus_write_data_ap (swd_ctx_t * swdctx, swd_operation_t operation, int * data)

Perform (MOSI) data write with automatic parity calculation.

Parameters

**swdctx* swd context pointer.

operation type of action to perform on generated command.

**data* payload value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.2.5.16 `int swd_bus_write_data_p (swd_ctx_t * swdctx, swd_operation_t operation, int * data, char * parity)`

Perform (MOSI) data write with provided parity value.

Parameters

- *swdctx* swd context pointer.
- operation* type of action to perform on generated command.
- *data* payload value pointer.
- *parity* payload parity value pointer.

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.2.5.17 `int swd_bus_write_request (swd_ctx_t * swdctx, swd_operation_t operation, char * APnDP, char * RnW, char * addr)`

Perform Request.

Parameters

- *swdctx* swd context pointer.
- operation* type of action to perform with generated request.
- *APnDP* AccessPort (high) or DebugPort (low) access value pointer.
- *RnW* Read (high) or Write (low) access value pointer.
- *addr* target register address value pointer.

Returns

number of commands processed, or SWD_ERROR_CODE on failure.

5.2.5.18 `int swd_cmd_enqueue (swd_ctx_t * swdctx, swd_cmd_t * cmd)`

Append selected command to a context's command queue.

Parameters

- *swdctx* swd context pointer containing the command queue.
- *cmd* command to be appended to the context's command queue.

Returns

number of elements appended or SWD_ERROR_CODE on failure.

5.2.5.19 int swd_cmd_enqueue_miso_ack (swd_ctx_t * swdctx, char ** ack)

Append queue with ACK read.

Parameters

**swdctx* swd context pointer.

**ack* packet value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.20 int swd_cmd_enqueue_miso_data (swd_ctx_t * swdctx, int ** data)

Append command queue with data read.

Parameters

**swdctx* swd context pointer.

**data* data pointer.

Returns

of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.21 int swd_cmd_enqueue_miso_data_p (swd_ctx_t * swdctx, int ** data, char ** parity)

Append command queue with data and parity read.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

**parity* parity value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.2.5.22 int swd_cmd_enqueue_miso_n_data_p (swd_ctx_t * swdctx, int ** data, char ** parity, int count)

Append command queue with series of data and parity read.

Parameters

**swdctx* swd context pointer.

***data* data value array pointer.

***parity* parity value array pointer.

count number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.2.5.23 int swd_cmd_enqueue_miso_nbit (swd_ctx_t * swdctx, char ** data, int count)

Append command queue with bus binary read bit-by-bit operation.

This function will append command to the queue for each bit, and store one bit into single char array element, so read is not constrained to 8 bits. On error memory is released and appropriate error code is returned. Important: Memory pointed by *data must be allocated prior call!

Parameters

- *swdctx* swd context pointer.
- **data* allocated data array to write result into.
- count* number of bits to read (also the **data size).

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.2.5.24 int swd_cmd_enqueue_miso_parity (swd_ctx_t * swdctx, char ** parity)

Append command queue with parity bit read.

Parameters

- *swdctx* swd context pointer.
- *parity* parity value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.25 int swd_cmd_enqueue_miso_trn (swd_ctx_t * swdctx)

Append command queue with Turnaround activating MISO mode.

Parameters

- *swdctx* swd context pointer.

Returns

return number of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.26 int swd_cmd_enqueue_mosi_control (swd_ctx_t * swdctx, char * ctlmsg, int len)

Append command queue with len-octet size control sequence.

This control sequence can be used for instance to send payload of packets switching DAP between JTAG and SWD mode.

Parameters

- *swdctx* swd context pointer.

**ctlmsg* control message array pointer.

len number of elements to send from **ctlmsg*.

Returns

number of elements appended (*len*), or SWD_ERROR_CODE on failure.

5.2.5.27 int swd_cmd_enqueue_mosi_dap_reset (swd_ctx_t * swdctx)

Append command queue with SW-DP-RESET sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.28 int swd_cmd_enqueue_mosi_data (swd_ctx_t * swdctx, int * data)

Append command queue with data and parity write.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.29 int swd_cmd_enqueue_mosi_data_ap (swd_ctx_t * swdctx, int * data)

Append command queue with data and automatic parity write.

Parameters

**swdctx* swd context pointer.

**data* data value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.2.5.30 int swd_cmd_enqueue_mosi_data_p (swd_ctx_t * *swdctx*, int * *data*, char * *parity*)

Append command queue with data and provided parity write.

Parameters

- *swdctx* swd context pointer.
- *data* data value pointer.
- *parity* parity value pointer.

Returns

number of elements appended (2), or SWD_ERROR_CODE on failure.

5.2.5.31 int swd_cmd_enqueue_mosi_idle (swd_ctx_t * *swdctx*)

Append command queue with idle sequence.

Parameters

- *swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.32 int swd_cmd_enqueue_mosi_jtag2swd (swd_ctx_t * *swdctx*)

Append command queue with JTAG-TO-SWD DAP-switch sequence.

Parameters

- *swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.33 int swd_cmd_enqueue_mosi_n_data_ap (swd_ctx_t * *swdctx*, int ** *data*, int *count*)

Append command queue with series of data and automatic parity writes.

Parameters

- *swdctx* swd context pointer.
- **data* data value array pointer.
- count* number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.2.5.34 `int swd_cmd_enqueue_mosi_n_data_p (swd_ctx_t * swdctx, int ** data, char ** parity, int count)`

Append command queue with series of data and provided parity writes.

Parameters

**swdctx* swd context pointer.
***data* data value array pointer.
***parity* parity value array pointer.
count number of (data+parity) elements to read.

Returns

number of elements appended (2*count), or SWD_ERROR_CODE on failure.

5.2.5.35 `int swd_cmd_enqueue_mosi_nbit (swd_ctx_t * swdctx, char * data, int count)`

Append command queue with bus binary write bit-by-bit operation.

This function will append command to the queue for each bit and store one bit into single char array element, so read is not constrained to 8 bits. On error memory is released and appropriate error code is returned. Important: Memory pointed by *data must be allocated prior call!

Parameters

**swdctx* swd context pointer.
***data* allocated data array to write result into.
count number of bits to read (also the **data size).

Returns

number of elements processed, or SWD_ERROR_CODE on failure.

5.2.5.36 `int swd_cmd_enqueue_mosi_parity (swd_ctx_t * swdctx, char * parity)`

Append command queue with parity bit write.

Parameters

**swdctx* swd context pointer.
**parity* parity value pointer.

Returns

number of elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.37 int swd_cmd_enqueue_mosi_request (swd_ctx_t * *swdctx*, char * *request*)

Appends command queue with SWD Request packet header.

Note that contents is not validated, so bad request can be sent as well.

Parameters

**swdctx* swd context pointer.

**request* pointer to the 8-bit request payload.

Returns

return number elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.38 int swd_cmd_enqueue_mosi_swd2jtag (swd_ctx_t * *swdctx*)

Append command queue with SWD-TO-JTAG DAP-switch sequence.

Parameters

**swdctx* swd context pointer.

Returns

number of elements appended, or SWD_ERROR_CODE on failure.

5.2.5.39 int swd_cmd_enqueue_mosi_trn (swd_ctx_t * *swdctx*)

Append command queue with Turnaround activating MOSI mode.

Parameters

**swdctx* swd context pointer.

Returns

return number elements appended (1), or SWD_ERROR_CODE on failure.

5.2.5.40 int swd_cmdq_append (swd_cmd_t * *cmdq*, swd_cmd_t * *cmd*)

Append element pointed by **cmd* at the end of the quque pointed by **cmdq*.

After this operation queue will be pointed by appended element (ie. last element added becomes actual quque pointer to show what was added recently).

Parameters

**cmdq* pointer to any element on command queue

**cmd* pointer to the command to be appended

Returns

number of appended elements (one), SWD_ERROR_CODE on failure

5.2.5.41 `swd_cmd_t* swd_cmdq_find_root (swd_cmd_t * cmdq)`

Find queue root (first element).

Parameters

**cmdq* pointer to any queue element

Returns

swd_cmd_t* pointer to the first element (root), NULL on failure

5.2.5.42 `swd_cmd_t* swd_cmdq_find_tail (swd_cmd_t * cmdq)`

Find queue tail (last element).

Parameters

**cmdq* pointer to any queue element

Returns

swd_cmd_t* pointer to the last element (tail), NULL on failure

5.2.5.43 `int swd_cmdq_flush (swd_ctx_t * swdctx, swd_operation_t operation)`

Flush command queue contents into interface driver.

Operation is specified by SWD_OPERATION and can be used to select how to flush the queue, ie. head-only, tail-only, one, all, etc.

Parameters

**swdctx* swd context pointer.

operation tells how to flush the queue.

Returns

number of commands transmitted, or SWD_ERROR_CODE on failure.

5.2.5.44 `int swd_cmdq_free (swd_cmd_t * cmdq)`

Free queue pointed by **cmdq* element.

Parameters

**cmdq* pointer to any element on command queue

Returns

number of elements destroyed, SWD_ERROR_CODE on failure

5.2.5.45 `int swd_cmdq_free_head (swd_cmd_t * cmdq)`

Free queue head up to *cmdq element.

Parameters

**cmdq* pointer to the element that becomes new queue root.

Returns

number of elements destroyed, or SWD_ERROR_CODE on failure.

5.2.5.46 `int swd_cmdq_free_tail (swd_cmd_t * cmdq)`

Free queue tail starting after *cmdq element.

Parameters

**cmdq* pointer to the last element on the new queue.

Returns

number of elements destroyed, or SWD_ERROR_CODE on failure.

5.2.5.47 `int swd_cmdq_init (swd_cmd_t * cmdq)`

Initialize new queue element in memory that becomes a queue root.

Parameters

**cmdq* pointer to the command queue element of type [swd_cmd_t](#)

Returns

SWD_OK on success, SWD_ERROR_CODE code on failure

5.2.5.48 `int swd_dap_detect (swd_ctx_t * swdctx, swd_operation_t operation, int ** idcode)`

Macro: Reset target DAP, select SW-DP, read out IDCODE.

This is the proper SW-DP initialization as stated by ARM Information Center.

Parameters

**swdctx* swd context pointer.

operation type (SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE).

Returns

Target's IDCODE, or error code on failure.

5.2.5.49 int swd_dap_reset (swd_ctx_t * swdctx, swd_operation_t operation)

Debug Access Port Reset sends 50 CLK with TMS high that brings both SW-DP and JTAG-DP into reset state.

Parameters

**swdctx* swd context pointer.

operation type (SWD_OPERATION_ENQUEUE or SWD_OPERATION_EXECUTE).

Returns

number of elements processed or SWD_ERROR_CODE code on failure.

5.2.5.50 int swd_dap_select (swd_ctx_t * swdctx, swd_operation_t operation)

Activate SW-DP by sending out RESET and JTAG-TO-SWD sequence on SWDIOTMS line.

Parameters

**swdctx* swd context.

Returns

number of control bytes executed, or error code on failure.

5.2.5.51 int swd_deinit (swd_ctx_t * swdctx)

De-initialize selected swd context and its command queue.

Parameters

**swdctx* swd context pointer.

Returns

number of elements freed, or SWD_ERROR_CODE on failure.

5.2.5.52 int swd_deinit_cmdq (swd_ctx_t * swdctx)

De-initialize command queue and free its memory on selected swd context.

Parameters

**swdctx* swd context pointer.

Returns

number of commands freed, or SWD_ERROR_CODE on failure.

5.2.5.53 `int swd_deinit_ctx (swd_ctx_t * swdctx)`

De-initialize selected swd context and free its memory.

Note: This function will not free command queue for selected context!

Parameters

**swdctx* swd context pointer.

Returns

SWD_OK on success, SWD_ERROR_CODE on failure.

5.2.5.54 `int swd_dp_read_idcode (swd_ctx_t * swdctx, swd_operation_t operation, int ** idcode)`

Macro: Read out IDCODE register and return its value on function return.

Parameters

**swdctx* swd context pointer.

operation operation type.

Returns

Target's IDCODE value or code error on failure.

5.2.5.55 `int swd_drv_transmit (swd_ctx_t * swdctx, swd_cmd_t * cmd)`

Transmit selected command from the command queue to the interface driver.

Parameters

**swdctx* swd context pointer.

**cmd* pointer to the command to be sent.

Returns

number of commands transmitted (1), or SWD_ERROR_CODE on failure.

5.2.5.56 `swd_ctx_t* swd_init (void)`

LibSWD initialization routine.

It should be called prior any operation made with libswd. It initializes command queue and basic parameters for context that is returned as pointer.

Returns

pointer to the initialized swd context.

5.2.5.57 `int swd_log (swd_ctx_t * swdctx, swd_loglevel_t loglevel, char * msg, ...)`

Put a message into swd context log at specified verbosity level.

If specified message's log level is lower than actual context configuration, message will be omitted. Verbosity level increases from 0 (silent) to 4 (debug).

Parameters

**swdctx* swd context.

loglevel at which to put selected message.

**msg* message body with variable arguments as in "printf".

Returns

number of characters written or error code on failure.

5.2.5.58 `int swd_log_level_set (swd_ctx_t * swdctx, swd_loglevel_t loglevel)`

Change log level to increase or decrease verbosity level.

Parameters

**swdctx* swd context.

loglevel is the target verbosity level to be set.

Returns

SWD_OK on success or error code.