libswd

0.0.1

Generated by Doxygen 1.7.1

Fri Mar 4 2011 12:57:36

Contents

Chapter 1

Serial Wire Debug Open Library.

1.1 Introduction

Welcome to the source code documentation repository. LibSWD is an Open-Source framework to deal with with Serial Wire Debug. 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 to access Debug Access Port in ARM-Cortex's based devices. LibSWD provides both bitstream generation and high/low level bus operations. Every bus operation such as request, turnaround, acknowledge, data and parity packet is represented by a swd_cmd_t element that can extend command queue (a standard bidirectional queue) that later can be flushed into real hardware using simple set of interface-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 drivers that controls the interface interconnecting host and target. Such drivers are application specific therefore located in external file crafted for that application and its hardware.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 swd_ahbap_t Struct Reference

Most actual Advanced High Bandwidth Access Peripherial Bus Reisters.

`#include <libswd.h>`

Public Attributes

- 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 Peripherial Bus Reisters.

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
  < Payload data union.
  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**

- struct swd_cmd_t * next
  Pointer to the previous/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.

4.2.2 Member Data Documentation

4.2.2.1 char swd_cmd_t::TRNnMOSI

< Payload data union.

Holds/sets bus direction: MOSI when zero, MISO for other.

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.

- swd_swdp_t misoswdp

  Last known read from the SW-DP registers.

- swd_swdp_t mosiswdp

  Last known write to the SW-DP registers.

- swd_ahbap_t misoahbap

  Last known read from AHB-AP registers.

- swd_ahbap_t mosiahbap

  Last known write ti the AHB-AP registers.

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

Chapter 5

File Documentation

5.1 libswd.c File Reference

```
#include <libswd.h>
#include <urjtag/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 seruence.
- int swd_cmd_enqueue_mosi_dap_reset (swd_ctx_t *swdctx)
  Append command queue with SW-DP-RESET 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.
- 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.
- 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, char *data, int bits, int nLSBfirst)
- int **swd_drv_mosi_32** (swd_ctx_t *swdctx, int *data, int bits, int nLSBfirst)

- int **swd_drv_miso_8** (swd_ctx_t *swdctx, char *data, int bits, int nLSBfirst)
- int **swd_drv_miso_32** (swd_ctx_t *swdctx, 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_bus_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_mosi_request (swd_ctx_t *swdctx, swd_operation_t operation, char *APnDP, char *RnW, char *addr)

  Perform Request.

- int swd_miso_ack (swd_ctx_t *swdctx, swd_operation_t operation, char *ack)

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

- int swd_mosi_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int *data, char *parity)

  Perform (MOSI) data write with provided parity value.

- int swd_mosi_data_ap (swd_ctx_t *swdctx, swd_operation_t operation, int *data)

  Perform (MOSI) data write with automatic parity calculation.

- int swd_miso_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int *data, char *parity)

  Perform (MISO) data read.

- int swd_mosi_control (swd_ctx_t *swdctx, swd_operation_t operation, char *ctlmsg, int len)

  Write CONTROL byte to the Target's DAP.

- int swd_mosi_jtag2swd (swd_ctx_t *swdctx, swd_operation_t operation)

  Switch DAP into SW-DP.

- int swd_miso_idcode (swd_ctx_t *swdctx, swd_operation_t operation, int *idcode, char *ack, char *parity)

  Read target's IDCODE register value.

- 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_swj (swd_ctx_t *swdctx, swd_operation_t operation)

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

- int swd_dap_idcode (swd_ctx_t *swdctx, swd_operation_t operation)

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

- int swd_dap_detect (swd_ctx_t *swdctx, swd_operation_t operation)

  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.

- 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_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.

5.1.2.11 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.

5.1.2.12 int swd_bus_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.

5.1.2.13 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.

5.1.2.14 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.

5.1.2.15 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.

5.1.2.16 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.

5.1.2.17 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.

5.1.2.18 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 apropriate 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.

5.1.2.19 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.

5.1.2.20 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.

5.1.2.21 int swd_cmd_enqueue_mosi_control (swd_ctx_t ∗ *swdctx,* char ∗ *ctlmsg,* int *len*)

Append command queue with len-octet size control seruence.

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.

5.1.2.22 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.

5.1.2.23 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.

5.1.2.24 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.

5.1.2.25 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.

5.1.2.26 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.

5.1.2.27 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.

5.1.2.28 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.

5.1.2.29 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 apropriate 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.

5.1.2.30 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.

5.1.2.31 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.

5.1.2.32 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.

5.1.2.33 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.

5.1.2.34 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

5.1.2.35 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

5.1.2.36 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

5.1.2.37 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.

5.1.2.38 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

5.1.2.39 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.

5.1.2.40 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.

5.1.2.41 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

5.1.2.42 int swd_dap_detect (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

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.

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

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.

5.1.2.44 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.

5.1.2.45 int swd_dap_select_swj (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

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

Parameters

**swdctx* swd context.

Returns

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

5.1.2.46 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.

5.1.2.47 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.

5.1.2.48 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.

5.1.2.49 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.

5.1.2.50 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.

5.1.2.51 int swd_miso_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.

5.1.2.52 int swd_miso_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.

5.1.2.53 int swd_miso_idcode (swd_ctx_t ∗ *swdctx*, swd_operation_t *operation*, int ∗ *idcode*, char ∗ *ack*, char ∗ *parity*)

Read target's IDCODE register value.

Parameters

∗***swdctx*** swd context pointer.

operation type of action to perform (queue or execute).

∗***idcode*** resulting register value pointer.

∗***ack*** resulting acknowledge response value pointer.

∗***parity*** resulting data parity value pointer.

Returns

number of elements processed on the queue, or SWD_ERROR_CODE on failure.

5.1.2.54 int swd_mosi_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.

5.1.2.55 int swd_mosi_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.

5.1.2.56 int swd_mosi_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.

5.1.2.57 int swd_mosi_jtag2swd (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

Switch DAP into SW-DP.

According to ARM documentation target's DAP use JTAG transport by default and so JTAG-DP is active after power up. To use SWD user must perform predefined sequence on SWDIO/TMS lines, then read out the IDCODE to ensure proper SW-DP operation.

5.1.2.58 int swd_mosi_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 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 Peripherial Bus Reisters.

- 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_DSTKCMPCLR 1
  DSTKCMPCLR 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_OCDBGRSTREQ 26
  OCDBGRSTREQ bit number.
- #define SWD_CTRLSTAT_BITNUM_OCDBGRSTACK 27
  OCDBGRSTACK 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_TURNOUND_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: 0xE00FF000.
- #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 enum SWD_ERROR_CODE swd_error_code_t
  Status and Error Codes definitions.
- typedef enum SWD_LOGLEVEL swd_loglevel_t
  Logging Level Codes definition.
- typedef enum SWD_CMDTYPE swd_cmdtype_t
  SWD Command Codes definitions.
- typedef enum SWD_SHIFTDIR swd_shiftdir_t
  What is the shift direction LSB-first or MSB-first.

- typedef enum SWD_OPERATION swd_operation_t

  Command Queue operations codes.

- typedef struct swd_cmd_t swd_cmd_t

  SWD Command Element Structure.

Enumerations

- enum SWD_ERROR_CODE {

  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_QUEUEROOT = -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 {

  SWD_LOGLEVEL_MIN = 0, SWD_LOGLEVEL_SILENT = 0, SWD_LOGLEVEL_ERROR = 1, SWD_LOGLEVEL_WARNING = 2,

  SWD_LOGLEVEL_INFO = 3, SWD_LOGLEVEL_DEBUG = 4, **SWD_LOGLEVEL_MAX** = 4 }

  Logging Level Codes definition.

- enum SWD_CMDTYPE {

  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 { SWD_DIR_LSBFIRST = 0, SWD_DIR_MSBFIRST = 1 }

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

- enum SWD_OPERATION {

  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_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 seruence.

- int swd_cmd_enqueue_mosi_dap_reset (swd_ctx_t *swdctx)

  Append command queue with SW-DP-RESET 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_transmit (swd_ctx_t *swdctx, swd_cmd_t *cmd)

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

- 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, char *data, int bits, int nLSBfirst)
- int **swd_drv_mosi_32** (swd_ctx_t *swdctx, int *data, int bits, int nLSBfirst)
- int **swd_drv_miso_8** (swd_ctx_t *swdctx, char *data, int bits, int nLSBfirst)
- int **swd_drv_miso_32** (swd_ctx_t *swdctx, int *data, int bits, int nLSBfirst)
- int swd_cmdq_flush (swd_ctx_t *swdctx, swd_operation_t operation)

  Flush command queue contents into interface driver.

- int swd_mosi_request (swd_ctx_t *swdctx, swd_operation_t operation, char *APnDP, char *RnW, char *addr)

  Perform Request.

- int swd_miso_ack (swd_ctx_t *swdctx, swd_operation_t operation, char *ack)

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

- int swd_mosi_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int *data, char *parity)

  Perform (MOSI) data write with provided parity value.

- int swd_mosi_data_ap (swd_ctx_t *swdctx, swd_operation_t operation, int *data)

  Perform (MOSI) data write with automatic parity calculation.

- int swd_miso_data_p (swd_ctx_t *swdctx, swd_operation_t operation, int *data, char *parity)

  Perform (MISO) data read.

- int swd_mosi_control (swd_ctx_t *swdctx, swd_operation_t operation, char *ctlmsg, int len)

  Write CONTROL byte to the Target's DAP.

- int swd_mosi_jtag2swd (swd_ctx_t *swdctx, swd_operation_t operation)

  Switch DAP into SW-DP.

- int swd_miso_idcode (swd_ctx_t *swdctx, swd_operation_t operation, int *idcode, char *ack, char *parity)

  Read target's IDCODE register value.

- 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_swj (swd_ctx_t *swdctx, swd_operation_t operation)

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

- int swd_dap_idcode (swd_ctx_t *swdctx, swd_operation_t operation)

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

- int swd_dap_detect (swd_ctx_t *swdctx, swd_operation_t operation)

  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.

- 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.

- int **swd_drv_mosi_trn** (swd_ctx_t *swdctx, int clks)
- int **swd_drv_miso_trn** (swd_ctx_t *swdctx, int clks)

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.

5.2.3.2 typedef enum SWD_CMDTYPE 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.

5.2.3.3 typedef enum SWD_ERROR_CODE swd_error_code_t

Status and Error Codes definitions.

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

5.2.3.4 typedef enum SWD_LOGLEVEL swd_loglevel_t

Logging Level Codes definition.

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

5.2.3.5 typedef enum SWD_OPERATION swd_operation_t

Command Queue operations codes.

5.2.3.6 typedef enum SWD_SHIFTDIR swd_shiftdir_t

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

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

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).

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

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

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 Log errors only.

SWD_LOGLEVEL_WARNING Log warnings.

SWD_LOGLEVEL_INFO Log informational messages.

SWD_LOGLEVEL_DEBUG Log all including debug information.

5.2.4.5 enum SWD_OPERATION

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

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_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.

5.2.5.11 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.

5.2.5.12 int swd_bus_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.

5.2.5.13 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.

5.2.5.14 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.

5.2.5.15 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.

5.2.5.16 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.

5.2.5.17 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.

5.2.5.18 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 apropriate 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.

5.2.5.19 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.

5.2.5.20 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.

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

Append command queue with len-octet size control seruence.

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.

5.2.5.22 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.

5.2.5.23 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.

5.2.5.24 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.

5.2.5.25 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.

5.2.5.26 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.

5.2.5.27 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.

5.2.5.28 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.

5.2.5.29 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 apropriate 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.

5.2.5.30 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.

5.2.5.31 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.

5.2.5.32 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.

5.2.5.33 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.

5.2.5.34 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

5.2.5.35 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

5.2.5.36 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

5.2.5.37 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.

5.2.5.38 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

5.2.5.39 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.

5.2.5.40 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.

5.2.5.41 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

5.2.5.42 int swd_dap_detect (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

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.

5.2.5.43 int swd_dap_idcode (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

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.

5.2.5.44 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.

5.2.5.45 int swd_dap_select_swj (swd_ctx_t * *swdctx*, swd_operation_t *operation*)

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

Parameters

swdctx swd context.

Returns

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

5.2.5.46 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.

5.2.5.47 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.

5.2.5.48 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.

5.2.5.49 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.

5.2.5.50 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.

5.2.5.51 int swd_miso_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.

5.2.5.52 int swd_miso_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.

5.2.5.53 int swd_miso_idcode (swd_ctx_t ∗ *swdctx*, swd_operation_t *operation*, int ∗ *idcode*, char ∗ *ack*, char ∗ *parity*)

Read target's IDCODE register value.

Parameters

∗***swdctx*** swd context pointer.

operation type of action to perform (queue or execute).

∗***idcode*** resulting register value pointer.

∗***ack*** resulting acknowledge response value pointer.

∗***parity*** resulting data parity value pointer.

Returns

number of elements processed on the queue, or SWD_ERROR_CODE on failure.

5.2.5.54 int swd_mosi_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.

5.2.5.55 int swd_mosi_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.

5.2.5.56 int swd_mosi_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.

5.2.5.57 int swd_mosi_jtag2swd (swd_ctx_t * *swdctx,* swd_operation_t *operation*)

Switch DAP into SW-DP.

According to ARM documentation target's DAP use JTAG transport by default and so JTAG-DP is active after power up. To use SWD user must perform predefined sequence on SWDIO/TMS lines, then read out the IDCODE to ensure proper SW-DP operation.

5.2.5.58 int swd_mosi_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.