SPANSION S26HL512T TESTBENCH STRUCTURE

This document describes the structure of test environment that will be used to verify the functional behavior and timing of the Spansion S26HL512T Flash memory VITAL model.
Test environment has two main parts: Package, that generates abstract stimuli information and Testbench that interfaces directly to the model under test, driving logical stimuli and accepts its responses.

Table of Contents

1 Package ---- 3
1.1 Command data type ---- 3
1.2 TC selection ---- 6
1.3 Status data type ---- 6
1.4 Aux type ---- 11
1.5 Checker procedures ---- 11
2 Testbench ---- 14
2.1 Test Parameters ---- 15
2.2 Testbench drivers ---- 15
2.3 Testbench Checkers ---- 16
2.4 Testbench defaults ---- 16

1 Package

Package defines several data types and procedure Generate_TC. Primary functionality of that procedure is to generate command sequence for defined test cases. Sequence is passed to test bench as a **global variable.**

1.1 Command data type

```
TYPE cmd_rec IS
  RECORD
    cmd    : cmd_type;
    status : status_type;
    data_w : NATURAL;
    sect   : INTEGER RANGE -1 TO 287
    addr   : NATURAL;
    aux    : Aux_type;
    wtime  : time; --valid with wt cmd
  END RECORD;
```

Each command in the sequence is given with its code, status that should be verified (if command is read), write data and read/write location (sector, address). Also, there are additional parameters aux (auxiliary parameter needed for some commands) and wtime for timing control.

- **done** Indicates to the test bench that command sequence is over. That should be default command.
- **idle** Idle command is used to drive bus to an inactive state.
- **h_reset** Hardware reset, polarity and name of reset signal is DUT (Device Under Test) dependent. Additional parameter is reset pulse width. Default value is 200 ns.
- **autoboot_rd** Read from autoboot register. Verify that correct data have been read by comparing it with the value in the deafolt register
- **rd_burst** Used for reading burst data from memory array
- **rd_asp** Used to read ASP register
- **rd_vcr_0** Used to read Volatile Configuration Register 0 data from CR_ASO
- **rd_nvcr_0** Used to read Nonvolatile Configuration Register 0 data from CR_ASO
- **rd_vcr_1** Used to read Volatile Configuration Register 1 data from CR_ASO
- **rd_nvcr_1** Used to read Nonvolatile Configuration Register 1 data from CR_ASO
- **rd_status** Used to read Status Register data from SR_ASO
- **rd_sfdp** Used to read ID SFDP ASO
- **rd_intsr** Used to read Interrupt Status register
- **rd_intcr** Used to read Interrupt Configuration register
- **rd_crc** Used to read CRC Low and High registers
- **rd_eccsr** Used to read ECC status
- **rd_por** Used to read PORTime register
- **rd_ssr** Used to read SSR ASO
- **rd_password** Used to read Password ASO
- **rd_ppb** Used to read PPB ASO
- **rd_ppblb** Used to read PPB Lock bit
- **rd_dyb** Used to read DYB ASO
- **rd_ecc** Used to read ECC
- **pass_unlock_12** Used to unlock Password
- **rd** SPI read command, generates bus read command. Parameters for this command are read location(sector number and address within that sector), number of reads (byte_num) and STATUS to be verified by that read.

- **rd_4** SPI read command, generates bus read command. Parameters for this command are read location(sector number and address within that sector), number of reads (byte_num) and STATUS to be verified by that read. (4 Bytes Address)
- **fast_rd4** SPI read command, generates bus read command. Parameters for this command are read location(sector number and address within that sector), number of read (byte_num) and STATUS to be verified by that read. (4 Bytes Address)
- **read_JID** JEDEC Standard 1312, read ID command. Parameters for this command are number of reads (byte_num) and STATUS to be verified by that read.
- **read_SFDP** Read Jedec Serial Flash Discoverable parameters.
- **rdar_read** SPI Read Any Register command
- **w_autoboot** Autoboot Command
- **wrar** SPI Write Any Register command. Parameters are write address and write data.
- **w_reset** Reset command (soft reset) forces model fsm to default read flash array state. No parameters.
- **w_reset_FF** Reset command (soft reset) with different data part than w_reset
- **w_set_exit** Exit command for different ASOs
- **w_cycle** Two cycle Unlock command sequence
- **w_erase** First command cycle of the Sector erase command sequence
- **w_BC** Command for sector Blank Check (SA|555/33)
- **w_autosel** Activate ID SFDP mode.
- **w_data** Part of the Write to buffer sequence.
- **w_data_burst** Part of the Write to buffer sequence for burst write
- **w_enable** Write Enable command. No parameters.
- **w_disable** Write Disable command. No parameters.
- **w_status_read** Status Register Read entry command (SA|555/70)
- **w_status_clear** Status Register Clear command (SA|555/71)
- **w_nvcr_read_0** Nonvolatile Configuration Register 0 Read entry command
- **w_vcr_read_0** Volatile Configuration Register 0 Read entry command
- **w_nvcr_read_1** Nonvolatile Configuration Register 1 Read entry command
- **w_vcr_read_1** Volatile Configuration Register 1 Read entry command
- **w_ers_nvcr** Nonvolatile Configuration Register Erase command
- **w_vcr_pgm_0** Volatile Configuration Register 0 Program command
- **w_nvcr_pgm_0** Nonvolatile Configuration Register 0 Program command
- **w_vcr_pgm_1** Volatile Configuration Register 1 Program command
- **w_nvcr_pgm_1** Nonvolatile Configuration Register 1 Program command
- **w_buff_cnf** Confirm command for initiating write to buffer program operation (SA|555/ 25).
- **w_pwd_cnf** Confirm command for initiating confirmation of password unlock sequence
- **w_spi** Used for entering SPI mode command
- **w_ssr** Used for SSR ASO enter command
- **w_ssr_55** Used for SSR ASO enter command with different address part than w_ssr
- **w_crc** Used for CRC ASO enter command
- **w_asp** Used for accessing ASP register
- **w_password** Used for entering Password ASO
- **w_ees** Used for enetering Evaluate Erase Status mode
- **w_sfdp** Used for entering ID SFDP mode
- **w_sfdp_55** Used for entering ID SFDP mode with different address part than w_sfdp
- **w_ppb** Used for entering PPB ASO
- **w_ppb_lock** Used for entering PPB Lock bit mode
- **w_dyb** Used for entering DYB ASO
- **w_ecc** Used for entering ECC Status

- **w_pg_por** Used for programming PORTime register
- **w_rd_por** Used for reading PORTime register
- **w_prog** Used for setting program command for various modes
- **w_ld_intcr** Used for entering Interrupt Configuration register
- **w_rd_intcr** Used for reading Interrutp Configuration register
- **w_ld_intsr** Used for loading Interrupt Status register
- **w_rd_intsr** Used for reading Interrupt Status register
- **w_buf_abort** Used to abort Buffer programming
- **w_aso_exit** Used to exit certain ASOs
- **w_chip_ers** Used to set Chip Erase command
- **w_ers_susp** Used to set Erase Suspend command
- **w_prog_susp** Used to set Program Suspend command
- **w_ers_resume** Used to set Erase Resume command
- **w_prog_resume** Used to set Program Resume command
- **w_atb_enter** Used for SPI entering AUTOBOOT mode
- **w_atb_pg** Used for programming AUTOBOOT register
- **w_ricrc** Used for entering Interface CRC read
- **w_eccr** Used for entering ECCR
- **w_mts** Measure Temperature Command
- **hfl_autoboot** HyperFlash AUTOBOOT mode
- **clock** Used for setting the clock frequency
- **wt** Wt command will wait for a time period specified with a command.wtime parameter.
- **rst** Software Reset command (99h). No parameters.
- **reset_en** Reset Enable command. No parameters
- **ees** Evaluate Erase Status command. No parameters
- **set_wl** Set Wrap Length command.
- **read_SR1** Read from Status Register 1 command. Parameters are number of read (byte_num) and STATUS to be verified by that read.
- **read_SR2** Read from Status Register 2 command. Parameters are number of read (byte_num) and STATUS to be verified by that read.
- **clr_sr** Clear status register bits
- **wrar** Write Any Register command. Parameters are write address and write data.
- **dp_down** Deep Power Down command.
- **sector_erase_4** Sector Erase command. Parameter for this command is sector selected for erase location (sector) . Additionally STATUS parameter controls how sector erase affects the default Testbench memory. (4 Bytes Address)
- **p4_erase_4** ParameterSector Erase command. Parameter for this command is sector selected for erase location (sector) . Additionally STATUS parameter controls how sector erase affects the default Testbench memory (4 Bytes Address).
- **bulk_erase_60** Bulk Erase command (60h). STATUS parameter controls how chip erase affects the default test bench memory.
- **bulk_erase_C7** Bulk Erase command (C7h). STATUS parameter controls how chip erase affects the default test bench memory.
- **ers_susp_75** Write Erase Suspend command (75h). No parameters.
- **ers_susp_85** Write Erase Suspend command (85h). No parameters.
- **ers_resume_7a** Write Erase Resume command (7Ah). No parameters.
- **ers_resume_8a** Write Erase Resume command (8Ah). No parameters.
- **pg_prog4** Page Program command. Parameters for this command are write data (data), write location (sector & address) and number of locations to program. Additionally STATUS parameter controls how current write affects the default Testbench

memory. (4 Bytes Address)

- **prg_susp_75** Write Program Suspend command (75h). No parameters.
- **prg_susp_85** Write Program Suspend command (85h). No parameters.
- **prg_resume _7a** Write Program Resume command (7A). No parameters.
- **prg_resume_8a** Write Program Resume command (8A). No parameters.
- **otp_prog** OTP Program command. Parameters for this command are write data (data), write location (sector & address) and number of locations to program. Additionally STATUS parameter controls how current write affects the default estbench memory.
- **otp_read** Read OTP command. Parameters for this command are number of reads (byte_num) and STATUS to be verified by that read.
- **ppbl_reg_rd** Read from PPB Lock Register command
- **w_ppbl_reg** Write to PPB Lock Register command
- **ppbacc_rd4** Read PPB Access Register command (4 Bytes Address)
- **w_ppb4** Write to PPB Access Register command (4 Bytes Address)
- **ppb_ers** Erase PPB bits command. No parameters.
- **ppb_ers_addr** Erase PPB Address bits command. No parameters.
- **dybacc_rd4** Read DYB Access Register command (4 Bytes Address)
- **w_dyb4** Write to DYB Access Register command (4 Bytes Address)
- **ecc_read** Read ECC Register Command
- **dicr_rd** Read DICR command

1.2 TC selection

*Which Testcase will be generated by **generate_TCprocedure**, is decided in **Pick_TC procedure.** Test cases are selected sequentially from TS1 to TS21 if parameter MODE has been set to DEFAULT_PROTECTION.. TS22 is run after Testbench generic parameter MODE has been set to PERSISTENT_PROTECTION. TS23 is run after Testbench generic parameter MODE has been set to PASSWORD_PROTECTION. TS24 is run after Testbench generic parameter MODE has been set to TEST_AUTOBOOT. TS25 is run after Testbench generic parameter MODE has been set to TEST_SFDP. Tests from TS26 and TS42 are run after Testbench generic parameter MODE has been set to SPI_PROTECTION. TS42 is run after Testbench generic parameter MODE has been set to TEST_SPI_DPD. Information about the number of testcases in each testseries is stored in constant array **tc.***

```
--number of testcases per testseries
TYPE tc_list IS ARRAY(1 TO 41) OF NATURAL;
CONSTANT tc : tc_list :=
--0                 1                   2                   3                   4
--1-2-3-4-5-6-7-8-9-0-1-2-3-4-5-6-7-8-9-0-1-2-3-4-5-6-7-8-9-0-1-2-3-4-5-6-7-8-9-0-1-2
 (1,4,2,6,5,5,8,9,2,2,2,2,3,2,1,3,1,3,1,2,1,1,6,1,1,2,2,1,5,1,1,2,1,1,1,1,1,3,1,1,2,2);-
```

1.3 Status data type

```
-- list of data/status to be verified by checker
   TYPE status_type IS (none,
                        linear,
                        cont,
                        rd_HiZ,
```

readX,
read_4,
read_fast_4,
read_autoboot,
read_rdar,
rd_U,
read_otp,
rd_JID,
rd_SFDP,
rd_SR1,
rd_SR2,
read_ppbl,
read_ppbar_4,
read_ecc,
read_dybar_4,
rd_ppblock_0,
rd_ppblock_1,
rd_wip_0,
rd_wip_1,
rd_wel_0,
rd_wel_1,
erase_succ,
erase_nosucc,
pgm_succ,
pgm_nosucc,
rd_ps_0,
rd_ps_1,
rd_es_0,
rd_es_1,
chk_sts_1,
chk_sts_0,
cont_ers_success,
cont_ers_fail,
cont_prog_fail,
cont_prog_success,
linear_ers_success,
linear_ers_fail,
linear_prog_fail,
linear_prog_success,
cont_device_busy,
cont_device_ready,
linear_device_busy,
linear_device_ready,
prmsec_erase_active,
erase_active,
erase_na,
wr2buff,
buff_wr_busy,
ssr_data,
cont_buff_abort,
linear_buff_abort,
cont_buff_not_abort,
linear_buff_not_abort,
cont_ers_susp,
linear_ers_susp,

```
cont_prog_susp,
linear_prog_susp,
cont_ers_not_susp,
linear_ers_not_susp,
cont_prog_not_susp,
linear_prog_not_susp,
cont_ees_not_success,
linear_ees_not_success,
cont_ees_success,
linear_ees_success,
cont_sect_lock,
linear_sect_lock,
cont_sect_not_lock,
linear_sect_not_lock,
asp_reg,
port_reg,
lock_bit,
PPBs,
PPB_pg,
PPB_ers,
DYBs,
DYB_pg,
passwd,
crc_bl,
crc_el,
crc_rd,
crc_susp
interrupt_cr,
interrupt_sr,
err,
wrap
);
```

STATUS parameter is used to tell the checker what is the expected behavior of the model under test.

- **none** - Default value. Nothing to verify. When used with w_data command, write data is written in the default memory array
- **linear** - Use with rd_burst command to specify linear burst read with wrap around.
- **cont** - Use with rd_burst command to specify continous burst read.
- **rd_HiZ** - Output value should be all HiZ.
- **readX** - Output value should be all X. Read location data integrity violated.
- **read_4** - Read from flash array. Verify that correct data have been read by comparing it with the value in the default memory.
- **read_fast_4** - Read from flash array. Verify that correct data have been read by comparing it with the value in the default memory.
- **read_autoboot** - Read from autoboot register. Verify that correct data have been read by comparing it with the value in the default register.
- **read_rdar** - Read from register selected by defined address. Verify that correct data have been read by comparing it with the value in the default register.
- **cont_sect_lock-** Use with rd_status command to read Device Ready Bit of the Status Register to verify that sector is locked or not locked (continuous burst read).
- **rd_U** Output value should be undefined.
- **read_otp** read from OTP memory. Verify that correct data have been read by comparing it with the value in the default OTP memory.
- **rd_JID** read Manufacturer ID and Device ID (JEDEC Standard 1312). Verify that correct ID have been read by comparing it with the device ID value .

- **rd_JQID** Quad read Manufacturer ID and Device ID (JEDEC Standard 1312). Verify that correct ID have been read by comparing it with the device ID value.
- **rd_SFDP** Read Jedec Serial Flash Discoverable parameters.
- **rd_SR1** read from status register 1. Verify that correct data have been read by comparing it with the value in the default register.
- **rd_SR2** read from status register 2. Verify that correct data have been read by comparing it with the value in the default register.
- **read_ppbl,**- Used with ppbl_reg_rd command to check read PPB Lock Register
- **read_ppbar_4,**- Used with ppbacc_rd4 command check read PPB Access Registers
- **read_ecc,-** Used with ecc_read command to check read from ECC Register
- **read_dybar_4,-** Used with dybacc_rd4 command to check read from Dynamic Protection Bits
- **rd_ppblock_0,-** Indicate that PPB array may be programed or erased
- **rd_ppblock_1,** - Indicate that PPB array is protected
- **rd_wip_0** - Read WIP bit to verify that Write Status, Program or Erase cycle is not in progress.
- **rd_wip_1** - Read WIP bit to verify that Write Status, Program or Erase cycle is in progress.
- **rd_wel_0** - Read WEL bit to verify that Write Status, Program or Erase cycle is enabled.
- **rd_wel_1** - Read WEL bit to verify that Write Status, Program or Erase cycle is completed.
- **erase_succ** - Read E_ERR bit to verify if Erase operation is finished without error.
- **erase_nosucc-** Read E_ERR bit to verify if Erase operation is finished with error.
- **pgm_succ** - Read P_ERR bit to verify if Program operation is finished without error.
- **pgm_nosucc-** Read P_ERR bit to verify if Program operation is finished with error.
- **rd_ps_0** - Read PS bit to verify that Program Suspend is not in progress.
- **rd_ps_1** - Read PS bit to verify that Program Suspend is in progress.
- **rd_es_0** - Read ES bit to verify that Erase Suspend is not in progress.
- **rd_es_1-** Read ES bit to verify that Erase Suspend is in progress.
- **chk_sts_1** - Read ESTAT bit to verify that previously issued Erase was successful.
- **chk_sts_0** - Read ESTAT bit to verify that previously issued Erase was unsuccessful.
- **cont_ers_success**- Use with rd_burst command to read Erase Status Bit of the Status Register in no busy state to verify that the most recent erase operation was successful (continuous burst read).
- **cont_ers_fail**- Use with rd_burst command to read Erase Status Bit of the Status Register in no busy state to verify that the most recent erase operation was not successful (continuous burst read).
- **cont_prog_fail**- Use with rd_burst command to read Program Status Bit of the Status Register in no busy state to verify that the most recent program operation was successful (continuous burst read).
- **cont_prog_success**- Use with rd_burst command to read Program Status Bit of the Status Register in no busy state to verify that the most recent program operation was not successful (continuous burst read).
- **linear_ers_success**- Use with rd_burst command to read Erase Status Bit of the Status Register in no busy state to verify that the most recent erase operation was successful (linear burst read).
- **linear_ers_fail**- Use with rd_burst command to read Erase Status Bit of the Status Register in no busy state to verify that the most recent erase operation was not successful (linear burst read).
- **linear_prog_fail**- Use with rd_burst command to read Program Status Bit of the Status Register in no busy state to verify that the most recent program operation was successful (linear burst read).
- **linear_prog_success**- Use with rd_burst command to read Program Status Bit of the Status Register in no busy state to verify that the most recent program operation was not successful (linear burst read).
- **cont_device_busy**- Use with rd_burst command to read Device Ready Bit of the Status Register to verify that embedded programming or erase or sector blank check algorithm is in-progress (continuous burst read).
- **cont_device_ready**- Use with rd_burst command to read Device Ready Bit of the Status Register to verify that embedded programming or erase or sector blank check algorithm is not active (continuous burst read).
- **linear_device_busy**- Use with rd_burst command to read Device Ready Bit of the Status Register to verify that embedded programming or erase or sector blank check algorithm is in-progress (linear burst read).
- **linear_device_ready**- Use with rd_burst command to read Device Ready Bit of the Status Register to verify that embedded programming or erase or sector blank check algorithm is not active (linear burst read).

- **cont_buff_abort** - Used with rd_status command to read Write Buffer Abort status bit in Status register in continuous mode
- **linear_buff_abort**- Used with rd_status command to read Write Buffer Abort status bit in Status register in linear mode
- **cont_buff_not_abort**- Used with rd_status command to read Write Buffer Abort status bit in Status register in continuous mode
- **linear_buff_not_abort**- Used with rd_status command to read Write Buffer Abort status bit in Status register in linear mode
- **cont_ers_susp**- Used with rd_status command to read Erase Suspend status bit in Status register in continuous mode
- **linear_ers_susp**- Used with rd_status command to read Erase Suspend status bit in Status register in linear mode
- **cont_prog_susp**- Used with rd_status command to read Program Suspend status bit in Status register in continuous mode
- **linear_prog_susp** - Used with rd_status command to read Program Suspend status bit in Status register in linear mode
- **cont_ers_not_susp**- Used with rd_status command to read Erase Suspend status bit in Status register in continuous mode
- **linear_ers_not_susp**- Used with rd_status command to read Erase Suspend status bit in Status register in linear mode
- **cont_prog_not_susp**- Used with rd_status command to read Program Suspend status bit in Status register in continuous mode
- **linear_prog_not_susp**- Used with rd_status command to read Program Suspend status bit in Status register in linear mode
- **cont_ees_not_success** - Used in rd_status command to read Sector Erase status bit in Status register in continuous mode
- **linear_ees_not_success**- Used in rd_status command to read Sector Erase status bit in Status register in linear mode
- **cont_ees_success**- Used in rd_status command to read Sector Erase status bit in Status register in continuous mode
- **linear_ees_success**- Used in rd_status command to read Sector Erase status bit in Status register in continuous mode
- **erase_active** - embedded erase algorithm active. Used in w_data command to mark that Erase command will be executed normally (normal verification).
- **erase_na**- embedded erase algorithm not active. Used in w_data command to mark that Erase command will not be executed (verification is not accepted).
- **prmsec_erase_active** - parameter sector erase active. Used in w_data command to mark that Erase command will be executed normally (normal verification).
- **wr2buff**- used with w_data to indicate Write Buffer Entry phase and Write Buffer Count phase
- **buff_wr_busy**- used with w_data to indicate Write to Buffer Load phase
- **ssr_data-** write to SSR ASO. Used in w_data command to mark that data is written in SSR ASO
- **cont_sect_lock**- Used to verify if sector is locked in continuous mode
- **linear_sect_lock**- Used to verify if sector is locked in linear mode
- **cont_sect_not_lock**- Used to verify if sector is not locked in continuous mode
- **linear_sect_not_lock**- Used to verify if sector is not locked in linear mode
- **asp_reg**- Used in w_data command to indicate that data should be written in ASP register
- **port_reg**- Used in w_data command to indicate that data should be written in POR Time register
- **lock_bit**- Used to verify value of PPB Lock bit
- **PPBs** - Used to verify value of PPB bit
- **PPB_pg** Used in w_data command to indicate that data should be written in PPB protection register
- **PPB_ers**- Used in w_data command to indicate that PPB protection register should be erased
- **DYBs** - Used to verify value of DYB bit
- **DYB_pg** - Used in w_data command to indicate that DYB protection register should be programmed
- **passwd** - Used in w_data command to indicate that passwod is programmed
- **crc_bl** - Used in w_data command to establish start address for CRC calculation

- **crc_el** - Used in w_data command to establish end address and to start CRC calculation
- **crc_rd** - Used in w_data command to indicate that it is a read CRC enter command
- **crc_susp**– Used with rd_status command to read CRC Suspend status bit in Status register
- **interrupt_sr** - Used in w_data command to indicate that Interrupt Status register should be programmed
- **interrupt_cr** - Used in w_data command to indicate that Interrupt Configuration register should be programmed
- **err** - Used only with w_data to indicate that write data should not be written in default memory because cycle is part of command sequence or that current write command will not be accepted
- **wrap** - Used with rd_burst to test memory wraping feature

1.4 Auxtype

```
TYPE Aux_type IS ( valid,     -- valid operation
                   disable,   -- if used in write, ROE is asserted
                   violate    -- used to specify Write signal pulse width
                 );
```

AUX parameter is used to transfer some additional information, about current instruction, to testbench.
valid is the default value.
disable value is used with IDLE command to enable ROE assertion.
violate is used in command write cycle to control pulse width in order to test pulse protection

1.5 Checker procedures

Package has 46 checker procedures that are called by testbench to verify model's responses. Those procedures accept model's outputs (DQ and RDWS signals) and default memory contents (flash memory array, Autoselect codes, Status Register or Configuration Register parameters depending on STATUS command parameter). Procedures are defined as follows:

```
PROCEDURE Check_read_SPI(
    DQ : IN std_logic_vector(7 downto 0);
    Data : IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_rdar(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_Z_SPI(
    DQ : IN std_logic;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_X_SPI(
    DQ : IN std_logic;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_autoboot(
    DQ : IN std_logic_vector(15 downto 0);
    D_mem:IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_otp_read(
    DQ : IN std_logic_vector(15 downto 0);
    otp_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_sr1(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);
```

```
PROCEDURE Check_read_sr2(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_ppbl(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_ppbar(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_ecc(
     DQ : IN std_logic_vector(15 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_read_dybar(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_PPBLOCK_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_WIP_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_WEL_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_eers_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_epgm_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURECheck_PSUS_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURECheck_ESUS_bit(
    DQ : IN std_logic;
    sts : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_Z(
    DQ : IN std_logic_vector(7 downto 0);
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_X(
    DQ : IN std_logic_vector(7 downto 0);
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_Ers_Success(
```

```
        DQ   : IN std_logic;
        sts  : IN status_type;
        SIGNAL check_err: OUT std_logic);

    PROCEDURE Check_Prog_Success(
        DQ   : IN std_logic;
        sts  : IN status_type;
        SIGNAL check_err: OUT std_logic);

    PROCEDURE Check_nvcr_reg(
        DQ : IN std_logic_vector(15 downto 0);
        CR : IN NATURAL;
        SIGNAL check_err: OUT std_logic);

     PROCEDURE Check_vcr_reg(
        DQ : IN std_logic_vector(15 downto 0);
        CR : IN NATURAL;
        SIGNAL check_err: OUT std_logic);

    PROCEDURE Check_status_reg(
        DQ : IN std_logic_vector(15 downto 0);
        SR : IN NATURAL;
        SIGNAL check_err: OUT std_logic);

     PROCEDURE Check_read(
        DQ   : IN std_logic_vector(15 downto 0);
        Data : IN NATURAL;
        SIGNAL check_err: OUT std_logic);

     PROCEDURE Check_RDYBSY_bit(
        DQ : IN std_logic;
        sts : IN  status_type;
        SIGNAL check_err: OUT std_logic);

     PROCEDURE Check_RWDS(
        RDS      : IN std_logic;
        RDSValue : IN std_logic;
        SIGNAL check_err:  OUT std_logic);

    PROCEDURE Check_SecSi(
        DQ : IN std_logic_vector(15 downto 0);
        Data:  IN NATURAL;
        SIGNAL check_err: OUT std_logic);

     PROCEDURECheck_SFDP(
        DQ : IN std_logic_vector(15 downto 0);
        Data:  IN NATURAL;
        SIGNAL check_err: OUT std_logic);

     PROCEDURECheck_SFDP_sec_prot(
        DQ : IN std_logic;
        Data: IN std_logic;
        SIGNAL check_err: OUT std_logic);

    PROCEDURECheck_ASPReg(
        DQ : IN std_logic_vector(15 downto 0);
        REG : IN std_logic_vector(15 downto 0);
        SIGNAL check_err: OUT std_logic);

     PROCEDURECheck_PORTimeReg(
        DQ : IN std_logic_vector(15 downto 0);
        REG : IN std_logic_vector(15 downto 0);
        SIGNAL check_err: OUT std_logic);

     PROCEDURECheck_INTCR(
        DQ : IN std_logic_vector(15 downto 0);
        REG : IN std_logic_vector(15 downto 0);
        SIGNAL check_err: OUT std_logic);

    PROCEDURECheck_INTSR(
```

```
    DQ : IN std_logic_vector(15 downto 0);
    REG : IN std_logic_vector(15 downto 0);
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_Password(
    DQ      : IN std_logic_vector(15 downto 0);
    Password : IN std_logic_vector(15 downto 0);
    PassAddr : IN natural;
    ProtBit  : IN std_logic;
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_BitValue(
    Status     : IN status_type;
    DQ         : IN std_logic_vector(15 downto 0);
    ProtBit    : IN std_logic;
    SIGNAL check_err : OUT std_logic);

  PROCEDURE Check_Ers_Susp(
    DQ   : IN std_logic;
    sts  : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_Crc_Susp(
    DQ   : IN std_logic;
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_Prog_Susp(
    DQ   : IN std_logic;
    sts  : IN status_type;
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_Abort(
    DQ   : IN std_logic;
    sts  : IN status_type;
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_EES(
    DQ   : IN std_logic;
    sts  : IN status_type;
    SIGNAL check_err: OUT std_logic);

PROCEDURE Check_Sect_Locked(
    DQ   : IN std_logic;
    sts  : IN status_type;
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_CRC_Lo_reg(
    DQ : IN std_logic_vector(15 downto 0);
    Data: IN std_logic_vector(15 downto 0);
    SIGNAL check_err: OUT std_logic);

  PROCEDURE Check_CRC_Hi_reg(
    DQ : IN std_logic_vector(15 downto 0);
    Data: IN std_logic_vector(15 downto 0);
    SIGNAL check_err: OUT std_logic);
```

If checker procedures verify, expected output "READ OK" ty`pe of message is printed; if error is detected appropriate error message is generated and **check_err** signals is activated. Active value is '1'. This signal goes back to inactive value.
Since tests can be very long and with a large number of messages of both type, check_err signal can be used for monitoring and easy error location.
For more information see the STATUS parameter definition chapter.

2 Testbench

Testbench interprets abstract data stimuli provided by the generate_TC procedure and transforms it to actual bus cycles.

2.1 Test Parameters

Test environment can be used for verification of number of Spansion flash models of the S26HL family. There are several parameters for configuring Testbench to a particular model.

```
-----------------------------------------------------------------------
--memory configuration
-----------------------------------------------------------------------
CONSTANT MaxData       :NATURAL:= 16#FFFF#;
CONSTANT MemSize       :NATURAL:= 25#1FFFFFF#;
CONSTANT SecNum        :NATURAL:= 255;
CONSTANT SecSize       :NATURAL:= 18#1FFFF#;
CONSTANT PrmSecSize    :NATURAL:= 11#7FF#;
CONSTANT Lo_prm128KB_sa:NATURAL:= 15#4000#;
CONSTANT Hi_prm128KB_sa:NATURAL:= 16#FFFF#;
CONSTANT PageNum       :NATURAL:= 16384;
CONSTANT PageSize      :NATURAL:= 15;
CONSTANT HiAddrBit     :NATURAL:= 37;
CONSTANT AddrRANGE     :NATURAL:= 25#1FFFFFF#;
CONSTANT SecSi_size    :NATURAL:= 511;
CONSTANT WrBuffLength  :NATURAL:= 255;
-----------------------------------------------------------------------
```

MaxData parameter holds the maximum value to be stored in a memory cell.
MemSize parameter holds the address of the last location.
SecNum parameter holds the total number of flash memory sectors.
SecSize parameter holds the size of the flash memory sector.
PrmSecSize parameter holds the size of the Parameter Sector
Lo_prm128KB_sa parameter holds the value of starting address in low parameter sector
Hi_prm128KB_sa parameter holds the value of ending address in High parameter sector
PageSize parameter holds the size of the one Page.
PageNum parameter holds the total number of Pages within one sector.
HiAddrBit parameter is used to identify model memory address range.
WrBuffLength parameter holds the length of the Write Buffer.

2.2 Testbench drivers

Every command of the command sequence generated by **generate_TC** procedure is passed to the **cmd_dc** procedure. **cmd_dc** procedure breaks each command into one or more model specific bus cycles. Actual bus interfacing is done through **bus_cycle** procedure that drives one of defined bus cycles. Address and data are passed to this procedure from the command sequence or are generated in the cmd_dc procedure (for model's command cycles)

```
TYPE bus_type IS (bus_idle,
                  bus_standby, --CS# deasserted, others are don't care
                   bus_enable,  --CS# asserted, others deasserted
                  bus_reset,
                  bus_write,
                  bus_ca,
                  bus_dummy_byte,
                  bus_dummy_clock,
                  bus_mode_byte,
                  bus_read);
```

Defined bus cycles are:

bus_idle Bus idle cycle. All control signals are driven to their inactive values. If disable parameter is true Write Protect signal is asserted.

bus_enable Bus enable cycle. This cycle is usually followed by read or write cycle. In this cycle CS# is asserted low (device is selected), and all other inputs are inactive.

bus_reset Hardware reset cycle. RESET# signal is driven low for period specified by wt parameter (wtime from command sequence).

bus_write Bus write cycle. One byte is shifted into the device on DQ[7:0] signals. This cycle is usually followed by deselect cycle.

bus_ca Command/address cycle. Both Data and Command/Address information are transferred in a DDR fashion over the eight bit data bus. This bus cycle starts with the first three clock cycles providing the CAx (Command/Address) information indicating the transaction characteristics.

bus_dummy_byte Bus dummy_byte cycle. Dummy bytes are shifted in to the device on SI signal.

bus_dummy_clock Bus dummy_clock cycle. Dummy clock are present while SI signal has "don't care" value.

bus_mode_byte Bus mode_byte cycle. Mode byte is shifted in to the device on SI signal.

bus_read Bus read cycle. One or more data bytes are shifted out on DQ[7:0] signals. This cycle is usually followed by desel_read cycle.

2.3 Testbench Checkers

In order to verify that correct data is present on the SO line as the result of the read command, user have to specify what is expected to be read by that command. STATUS parameter is used for that.
According to that parameter **checker** process calls specific checker procedures from the package, passing to them parameters as SO signal, default memory contents, etc.

2.4 Testbench defaults

Default testbench process is to load default values into memory array. These values are used for model's read operation verification.
Memory array is initialized to all 0xFF value since that is the default flash memory state.
Memory array preload file name is <model name>.mem
Preload flash memory file is read and specified locations are filled with the values from the file.

Memory preload file is plain text file (ascii) formated in the following way:
/ - comment. Lines starting with / are skipped
@aaaaaa - <aaaaaa> stands for address within sector. Entries following this line will hold bytes to be loaded to locations starting from the one selected.
dd - <dd> is word to be written at aaaaaa++ location of every sector (aaaaa is incremented at every load)
Preload files should not have empty lines!

The Status Register is initialized to 0x00 value, all Status Register bits are 0.