FLASH TESTBENCH STRUCTURE

This document describes the structure of test environment that will be used to verify the functional behavior and timing of the S28HS512T Flash Memory VITAL model.
Test environment has two main parts: Testbench that instantiates the DUT, calls out the individual tests and interfaces directly to the model, driving logical stimuli and accepting its responses. Other part is the Package, that generates abstract stimuli information (Testcase vectors). Package also has Checker procedures that are called from the Testbench to verify model's response.

Table of Contents

1 Package----------3
  1.1 TC generation----------3
  1.2 Command data type----------3
  1.3 TC selection----------5
  1.4 Status data type----------7
  1.5 Aux type----------8
  1.6 Checker Procedures----------9
2 Testbench----------11
  2.1 Test Parameters----------11
  2.2 Testbench drivers----------12
  2.3 Testbench checkers----------13
  2.4 Testbench defaults----------13

1 Package

Package defines several data types and procedure. Primary functionality of the package is to generate command sequence for defined test case and to verify DUT response.
Command sequence is passed to Testbench as a **global variable.**

1.1 TC generation

Test case command sequence is generated by the Generate_TC procedure.
This procedure returns predefined command sequence of the selected test series/test case for the model selected.

```
PROCEDURE Generate_TC
  ( Model      : IN STRING;
    Series     : IN NATURAL RANGE 1 TO 45;
    TestCase   : IN NATURAL RANGE 1 TO 15;
    Sec_Arch   : IN BOOLEAN ;
    command_seq: OUT cmd_seq_type
  );
```

1.2 Command data type

```
TYPE cmd_rec IS
  RECORD
    cmd    : cmd_type;
    status : status_type;
    byte_num: NATURAL RANGE 0 TO 16#2000#;
    data4  : NATURAL;
    data3  : NATURAL;
    data2  : NATURAL;
    data1  : NATURAL;
    sect   : INTEGER RANGE 0 TO 287;
    addr   : INTEGER RANGE 0 TO 16#3FFFFF#;
    aux    : Aux_type;
    wtime  : time;
  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 (address). Also, there are additional parameters aux (auxiliary parameter needed for some commands) and wtime for timing control (for wait command).

- 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.
- w_enable — Write Enable command. No parameters.
- w_disable — Write Disable command. No parameters.
- reset_en — Reset Enable command. No parameters
- rst — Software Reset command (99h). No parameters.
- ees — Evaluate Erase Status command. No parameters
- 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 10 ns.
- wrar — Write Any Register command. Parameters are write address and write data.
- 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
- rd_4 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 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)
- quad_rd_4 Read command, generates bus quad 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)
- quad_high_rd read command, generates bus quad high performance 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.
- quad_high_rd_4 read command, generates bus quad high performance 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_ddr_rd read command, generates bus ddr 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.
- fast_ddr_rd4 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)
- dual_high_ddr_rd read command, generates bus dual high performance ddr 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.
- dual_high_ddr_rd4 read command, generates bus dual high performance ddr 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)
- quad_high_ddr_rd read command, generates bus quad high performance ddr 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.
- quad_high_ddr_rd_4 read command, generates bus quad high performance ddr 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)
- 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.
- res_dpd Release from DPD mode.
- 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 Parameter 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).
- 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_b0 Write Erase Suspend command (B0h). 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_b0 Write Program Suspend command (B0h). 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 Testbench memory.
- otp_read Read OTP command. Parameters for this command are number of reads (byte_num) and STATUS to be verified by that read.
- w_nvldr Program Nonvolatile Data Learning Patern command
- w_wvdlr Write Volatile DLP command
- asp_reg_rd Read from ASP Register command
- w_asp Write to ASP Register command
- psw_unlock Password unlock command. Parameter for this command is 64 bits long password (4 Bytes of data)
- 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.
- dybacc_rd4 Read DYB Access Register command (4 Bytes Address)
- ecc_read Read from ECC Register command
- rdar_read Read Any Register command
- w_crc Write to CRC Register command
- w_dyb4 Write to DYB Access Register command (4 Bytes Address)
- wt Wait for period specified by w_{TIME} parameter.
- inv_write Write less then 8 bits.
- seerc_rd Sector erase count read command
- rd_crc Read Interface CRC register value
- assert_cs Assert CSNeg
- jedec_reset Jedec Reset
- cs_ext_dpd CSNeg Low to Exit DPD

1.3 TC selection

Which Testcase will be generated by **generate_TC procedure**, is decided in **Pick_TC procedure.** Testcases are selected sequentially from TS1 to TS27. TS25 is run after Testbench generic parameter MODE has been set to PERSISTENT_PROTECTION. TS26 and TS27 are run after Testbench generic parameter MODE has been set to PASSWORD_PROTECTION. TS28 is run after Testbench generic parameter MODE has been set to PASSWORD_PROTECTION_OPI_SDR. TS29 is run after Testbench generic parameter MODE has been set to PASSWORD_PROTECTION_OPI_DDR. TS30 is run after Testbench generic parameter MODE has been set to TEST_SEERC_READ. And TS31 is run after Testbench generic parameter MODE has been set to TEST_JEDEC_RESET. 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 31) OF NATURAL;
CONSTANT tc : tc_list :=
--0                  1                   2                   3
--1-1-2-2-4-1-2-8-1-3-2-2-3-3-1-1-1-1-3-1-1-3-2-3-5-1-3-2-3-2-1-1
 (1,1,2,2,4,1,2,8,1,3,2,2,3,3,1,1,1,1,3,1,1,3,2,3,5,1,3,2,3,2,1,1);
```

Pick_TC procedure can be modified so the TC execution queue can be read from the file, or generated randomly.

1.4 Status data type

```
TYPE status_type IS ( none,
                      read_4,
                      read_fast_4,
                      rd_HiZ,
                      rd_U,
                      read_otp,
                      rd_JID,
                      rd_SFDP,
                      rd_SR1,
                      rd_SR2,
                      read_rdar,
                      read_dlp,
                      read_ppbl,
                      read_ppbar_4,
                      read_ecc,
                      read_dybar,
                      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,
                      err
                    );
```

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

- **none** default value. Nothing to verify.
- **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.
- **rd_HiZ** output value should be HiZ.
- **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_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_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.

- **read_dlp** read from dlp register. Verify that correct data have been read by comparing it with the value in the default register.
- **read_ppbl** read from ppb lock register. Verify that correct data have been read by comparing it with the value in the default register.
- **read_ppbar_4** read from ppb access register. Verify that correct data have been read by comparing it with the value in the default register.
- **read_ecc** read from ECC status register. Verify that correct data have been read by comparing it with the value in the default register.
- **read_dybar_4** read from dyb access register. Verify that correct data have been read by comparing it with the value in the default register.
- **rd_ppblock_0** Read PPB LOCK bit to verify that PPB array is protected.
- **rd_ppblock_1** Read PPB LOCK bit to verify that PPB array can be programmed or erased.
- **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.
- **err** used with write commands to indicate that write data should not be written in default memory because current write command will not be accepted.

1.5 Aux type

```
TYPE Aux_type IS (  valid,
                    violate,
                    clock_num,
                    break,
                    PowerUp
                 );
```

AUX parameter is used to transfer some additional information, about current instruction, to test bench.

- **valid** is the default value.

violate is used with IDLE command to assert/deassert W# (hardware write protection).

- **clock_num** is used with bus_code and bus_data_read cycle to drive CS# low for number of clock cycles that isn't multiple of eight.
- **break** is used in serial mode to break read before one of data is read, in parallel is used for mode byte read.
- **PowerUp** is used to mark that Power Up is in progress.

1.6 Checker Procedures

Package has 24 checker procedures that are called by testbench to verify model's responses. Those procedures accept model's outputs, registers bits and default memory contents.
Procedures are defined as follows:

```
PROCEDURE Check_read(
    DQ     : IN std_logic_vector(7 downto 0);
    DQ_reg0 : IN std_logic_vector(7 downto 0);
    DQ_reg1 : IN std_logic_vector(7 downto 0);
    DQ_reg2 : IN std_logic_vector(7 downto 0);
    DQ_reg3 : IN std_logic_vector(7 downto 0);
    D_mem  : IN NATURAL;
    DLP_reg : IN NATURAL;
    D_dlp_act: IN std_logic_vector(1 downto 0);
    SIGNAL check_err: OUT std_logic);

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

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

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

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

  PROCEDURE Check_read_JID(
    DQ          : IN std_logic_vector(7 downto 0);
    VDATA       : IN NATURAL;
    byte_no     : IN NATURAL;
    SIGNAL check_err: OUT std_logic);
  PROCEDURE Check_CRC_reg (
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);
  PROCEDURE Check_read_SFDP(
    DQ          : IN std_logic_vector(7 downto 0);
    VDATA       : 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_rdar(
    DQ : IN std_logic_vector(7 downto 0);
    D_mem: IN NATURAL;
    SIGNAL check_err: OUT std_logic);

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

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

PROCEDURE Check_read_autoboot(
  DQ : IN std_logic_vector(31 downto 0);
  D_mem: IN NATURAL;
  SIGNAL check_err: OUT std_logic);

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

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

PROCEDURE Check_read_pass_reg(
  DQ : IN std_logic_vector(63 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(7 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);
```

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

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

If checker procedures verify expected output "READ OK" type of massage is printed, and if error is detected appropriate error massage 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 flash models from the S25HS family. There are several parameters for configuring Testbench to a particular model.

```
CONSTANT MaxData        : NATURAL := 16#FF#;      --255;
CONSTANT MemSize        : NATURAL := 16#3FFFFFF#;
CONSTANT SecNumUni      : NATURAL := 255;
CONSTANT SecNumHyb      : NATURAL := 287;
CONSTANT SecSize4       : NATURAL := 16#FFF#;
CONSTANT SecSize256     : NATURAL := 16#3FFFF#;
CONSTANT PageNum512     : NATURAL := 16#1FFFF#;
CONSTANT PageNum256     : NATURAL := 16#3FFFF#;
CONSTANT AddrRANGE      : NATURAL := 16#3FFFFFF#;
CONSTANT HiAddrBit      : NATURAL := 31;
CONSTANT OTPSize        : NATURAL := 1023;
CONSTANT OTPLoAddr      : NATURAL := 16#000#;
CONSTANT OTPHiAddr      : NATURAL := 16#3FF#;
CONSTANT SFDPSize       : NATURAL := 16#223#;
CONSTANT SFDPLoAddr     : NATURAL := 16#0000#;
CONSTANT SFDPHiAddr     : NATURAL := 16#223#;
```

MaxData parameter holds the largest data that can be accepted.
MemSize parameter holds the size of flash memory.
SecNumUni parameter holds the total number of flash memory sectors for Uniform Arhitecture.
SecNumHyb parameter holds the total number of flash memory sectors for Hybrid Arhitecture.
SecSize4 parameter holds the size of flash memory parameter sectors.
SecSize256 parameter holds the size of flash memory sectors.
PageNum512 parameter holds the number of flash memory pages when PageSize is set to 512.
PageNum256 parameter holds the number of flash memory pages when PageSize is set to 256.
AddrRange parameter holds the highest address of flash memory.
HiAddrBit parameter that identify the MSB address bit (number of address bits minus one)
OTPSize parameter holds the size of OTP array.

OTPLoAddr parameter holds the lowest address of OTP array.
OTPHiAddr parameter holds the highest address of OTP array.
SFDPSize parameter holds the size of SFDP array.
SFDPLoAddr parameter holds the lowest address of SFDP array.
***SFDP*HiAddr** parameter holds the highest address of SFDP array.

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 six 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_select,       --CS# asseretd
                  bus_deselect,     --CS# deasserted after write
                  bus_desel_read,--CS# deasserted after read
                  bus_opcode,
                  bus_reset,
                  bus_address,
                  bus_dummy_byte,
                  bus_dummy_clock,
                  bus_mode_byte,
                  bus_data_read,
                  bus_data_write,
                  bus_st_delay,
                  bus_inv_write); -- write less then 8 bits
```

Defined bus cycles are:

bus_idle Bus idle cycle. CS# signal is driven high. If violate parameter is true Write Protect signal is asserted.

bus_select Bus select cycle. CS# signal is asserted. This cycle is usually followed by code cycle.

bus_deselect Bus deselect cycle. CS# signal is deasserted after Write, Program or Erase instruction.

bus_desel_read Bus deselect cycle. CS# signal is deasserted after any Read instruction.

bus_opcode Bus code cycle. One-byte Instruction code is shifted in to the device on SI signal. This cycle is usually followed by address cycle.

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

bus_address Bus address cycle. 3 byte/4 byte address is shifted in to the device on SI signal.

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_data_read Bus data_read cycle. One or more data bytes are shifted out on SO signal. This cycle is usually followed by desel_read cycle.

bus_data_write Bus data_write cycle. One to 512 data bytes are shifted in to the device on SI signal. This cycle is usually followed by deselect cycle.

bus_st_delay Bus start_delay cycle. This cycle is used in Autoboot mode.

bus_inv_write Write less then 8 bits

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.