UM0153
USER MANUAL

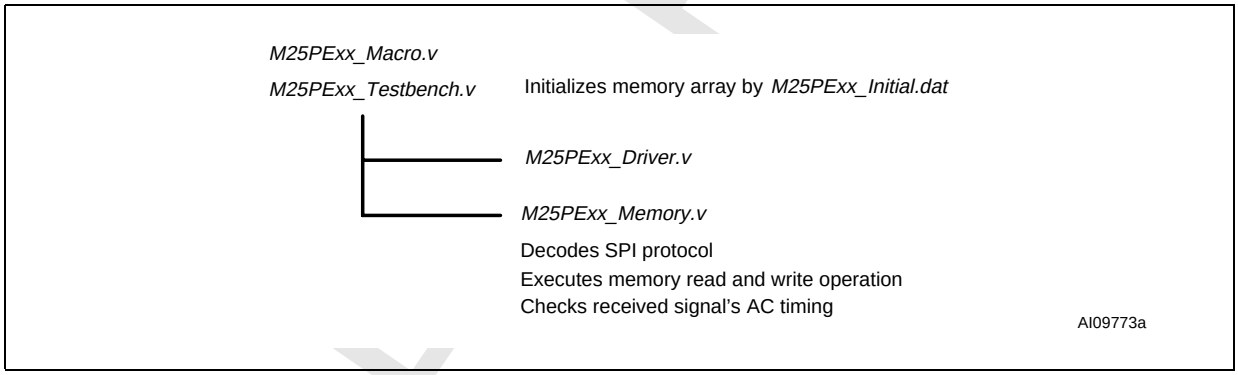
Verilog HDL Model for the M25PExx SPI Flash Pack

This Project gives a Verilog HDL behavioral model of the M25PExx family of SPI Serial Flash Memory devices. To give a more complete example of a Verilog HDL project, other Verilog HDL files are also provided.

The project is based on two blocks, the **M25PExx Memory** and **M25PExx Driver** blocks, gathered together under *M25PExx_Testbench.v* (as shown in Figure 1.).

- *M25PExx_Macro.v*
- *M25PExx_Testbench.v*: This initializes the content of the memory array, as specified in M25PExx_Initial.dat.
  - *M25PExx_Memory.v*: The Verilog simulation model of the M25PExx Flash memory simulates the behavior of the M25PExx memory. It decodes the SPI protocol, executes the memory Read and Write operations, and checks the AC timing of the received signal
  - *M25PExx_Driver.v*: This file defines the instructions that are to be transmitted to M25PExx memory.

Figure 1. Project architecture



When simulating a project, the following files must be compiled:

- *M25PExx _Macro.v* file: Defines parameters concerning the memory model configuration and AC characteristics.
  - Description of the "STOP" macro:

    "STOP = 0": Simulation continues and only displays a WARNING message when the model exceeds timing specifications.

    "STOP = 1": [Default] Simulation is STOPPED when the model exceeds timing specifications.
- *M25PExx _Memory.v* file: Describes the behavioral model of the M25PExx memory.
- *M25PExx _Testbench.v* file: Links the M25PExx memory and driver blocks.
- *M25PExx _Driver.v* file: Simulates an SPI bus master transmitting instructions to the *M25PExx_Memory.v* file.

Messages when running a simulation

When running a simulation, the M25PExx Verilog HDL model may send messages to display the status of the model, as commonly used in most Verilog HDL simulators:

Note: A note message contains non-critical information.

Warning: A warning message informs the user that the M25PExx Verilog HDL model is not properly driven (through the SPI bus) and that the SPI sequence is not compliant with the M25PExx specification.

Error: An error message informs the user that the M25PExx Verilog HDL model is not properly driven (through the SPI bus) and that the SPI sequence is not compliant with the M25PExx specification. The simulation will stop when this message is displayed.

How to use the initialization file

An included initialization file (*M25PExx_initial.dat*) defines the initialization content of the memory array before starting a simulation. When starting a simulation, the simulated memory array is automatically loaded with the contents of the M25PExx_initial.dat file.

The *M25PExx_initial.dat* file provided with this Verilog HDL model is filled with "FF" values but it can be easily replaced by another user's initialization file. In order to change the file, the user must change the parameter of system task *$readmemh* named *M25PExx_initial.dat* in the *M25PExx_testbench.v* file.

The Initial file format defined in this initialization file is:

- Each byte data is defined in Hexadecimal
- Each byte is packed into one line
- Each line (byte) ends with a <carriage return>

Driver file descriptions

The *M25PExx_driver.v* file only provides an example for driving the M25PExx SPI Flash memory model. It is easy to replace this file with the user's M25PExx memory access driver file. To replace the driver file, the user must change the *M25PExx_testbench.v* file, linking the user's new driver file to *M25PExx_memory.v* file for instantiation.

In this model, 17 tasks are defined in *M25PExx_driver.v* file. Every task corresponds to an instruction defined in M25PExx datasheets. These tasks are invoked in the *M25PExx_driver.v* file to generate drivers for memory access.

“WRITE_ENABLE”

This task generates a Write Enable instruction sequence and sets the WEL bit.

“WRITE_DISABLE”

This task generates a Write Disable instruction sequence and resets the WEL bit.

“READ_IDENTIFICATION”

This task generates a Read ID instruction sequence and reads out the Manufacturer ID and Device ID.

“READ_STATUS_REGISTER”

This task generates a Read Status Register instruction sequence and reads out the Status Register data.

“READ_DATA_BYTES”

This task generates a Read Data Bytes instruction sequence and reads out the data of the "n" bytes.

Format: *READ_DATA_BYTE (n, address)*

n: the number of data bytes that will be read out.

address: the start address for the Read Data Bytes instruction.

“READ_DATA_BYTES_FAST”

This task generates a Fast Read Data Bytes instruction sequence and reads out the data of the "n" bytes at a higher frequency.

Format: *READ_DATA_BYTE_FAST (n, address)*

n: the number of data bytes that will be read out.

address: the start address for the Fast Read Data Bytes instruction.

“PAGE_WRITE_SAME_DATA”

Format: *PAGE_WRITE_SAME_DATA (n, data, address)*

n: the number of data bytes that will be written in.

data: the value of data that will be written in.

address: the start address for the Page Write instruction.

This task generates a Page Write instruction sequence, erases the addressed page and writes "n" bytes data with the same value in memory beginning at the specified address.

When updating contiguous bytes in a page, optimized timings are obtained using a single Page Write sequence including all updated bytes versus using several Page Write sequences with each containing a just single byte. See datasheet and dedicated application note on programming times.

“PAGE_WRITE_DIFF_DATA”

Format: *PAGE_WRITE_DIFF_DATA (n, address)*

n: the number of data bytes that will be written in.

address: the start address for the Page Write instruction.

This task generates a Page Write instruction sequence, erases the addressed page and writes "n" bytes data with different value in memory beginning at the specified address.

This task should be used with a variable named *page_wr_buf*. Before invoking the *PAGE_WRITE_DIFF_DATA* task, it is necessary to assign a data value to the *page_wr_buf* variable, and then the *PAGE_WRITE_DIFF_DATA* task will write memory by data in *page_wr_buf*.

The specified format: page_wr_buf = {data1, data2… … … data N}; (N = 256 max.)

Example: Write 8 bytes data with different value in address 000000h~000007h.

```
page_wr_buf =
{8'haa,8'h55,8'hff,8'h00,8'h55,8'haa,8'h00,8'hff,8'hab,8'hcd,8
'hef,8'h01};
PAGE_WRITE_DIFF_DATA (12, 24'h000000);
//Wait for Page Write Cycle completed
```

When updating contiguous bytes in a page, optimized timings are obtained using a single Page Write sequence including all updated bytes versus using several Page Write sequences with each containing a just single byte. See datasheet and dedicated application note on programming times.

Note: Please ensure the number of data assigned to the page_wr_buf variable (N) is equal to the number of data bytes specified in the PAGE_WRITE_DIFF_DATA task.

“PAGE_PROGRAM_SAME_DATA”

Format: *PAGE_PROGRAM_SAME_DATA (n, data, address)*

n: the number of data bytes that will be programmed.

data: the value of data that will be programmed.

address: the start address for the Page Program instruction.

This task generates a Page Program instruction sequence, and programs "n" bytes data with the same value in memory beginning at the specified address.

Page Program instruction is optimized (time) for 256-byte sequences. Ensure that largest byte sequence is implemented. See datasheet and dedicated application note on programming time.

“PAGE_PROGRAM_DIFF_DATA”

Format: *PAGE_PROGRAM_DIFF_DATA (n, address)*

n: the number of data bytes that will be programmed in.

address: the destination address for the Page Program instruction.

This task generates a Page Program instruction sequence, and programs "n" bytes data with different value in memory beginning at the specified address.

This task should be used with a variable named *page_pg_buf* together. Before invoking the *PAGE_PROGRAM_DIFF_DATA* task, it is necessary to assign data value to the *page_pg_buf* variable, and then the *PAGE_PROGRAM_DIFF_DATA* task will program memory by data in *page_pg_buf*.

The specified format: page_pg_buf = {data1, data2… … … data N}; (N = 256 max.)

Example: Program 8 bytes data with different value in address 000000h~000007h.

```
page_pg_buf =
{8'haa,8'h55,8'hff,8'h00,8'h55,8'haa,8'h00,8'hff,8'hab,8'hcd,8
'hef,8'h01};
PAGE_PROGRAM_DIFF_DATA (12, 24'h000000);
//Wait for Page Program Cycle completed
```

Page Program instruction is optimized (time) for 256-byte sequences. Ensure that largest byte sequence is implemented. See datasheet and dedicated application note on programming time.

Note: Please ensure the number of data assigned to the variable page_pg_buf (N) is equal to the number of data bytes specified in task PAGE_PROGRAM_DIFF_DATA.

“PAGE_ERASE”

This task generates a Page Erase instruction sequence, and erases the addressed page.

Format: *PAGE_ERASE (address)*

address: the destination address for the Page Erase instruction.

“SECTOR_ERASE”

This task generates a Sector Erase instruction sequence, and erases the addressed sector.

Format: *SECTOR_ERASE (address)*

address: the destination address for the Sector Erase instruction.

“DEEP_POWER_DOWN”

This task generates a Deep Power-down instruction sequence.

“RELEASE_FROM_DEEP_POWER_DOWN"

This task generates a Release from the Deep Power-down instruction sequence.

"BULK_ERASE"(1)

This task generates a Bulk Erase instruction sequence and sets all bits to 1.

"WRITE_TO_LOCK_REGISTER"(1)

This task generates a Write to Lock Register instruction sequence and changes the bits in the Lock Register.

Format: *WRITE_TO_LOCK_REGISTER (address, data)*

address: address points to any location inside the targeted sector or sub-sector.

data: the value of data that will be written in the addressed lock register.

"READ_LOCK_REGISTER"(1)

This task generates a Read Lock Register instruction sequence and reads out the data of Lock Register.

Format: *READ_LOCK_REGISTER (address)*

address: address points to any location inside the concerned sector or sub-sector.

1. This task is available only when mentioned in Datasheet.

Revision history

Date	Revision	Changes
19-Jul-2005	1	Initial release.

CONFIDENTIALITY OBLIGATIONS:

This document contains sensitive information.
Its distribution is subject to the signature of an Non-Disclosure Agreement (NDA).
It is classified "**CONFIDENTIAL**"

At all times you should comply with the following security rules
(Refer to NDA for detailed obligations):

Do not copy or reproduce all or part of this document
Keep this document locked away

Further copies can be provided on a "need to know basis", please contact
your local ST sales office or go to the following request support web page: **http://www.st.com/xxx**