



Verilog model user manual

SPI EEPROM (M95xxx family)

(Version: 1.5, 6 Aug 2014)

Table of Contents

1. PROJECT ARCHITECTURE	3
a) Project	3
b) Source files	3
2. HOW TO DEFINE YOUR OWN PROJECT	4
a) Files to compile	4
b) How to use the define directives	5
c) How to use the initialization file	6
d) Messages when running a simulation	7
3. DRIVER FILES DESCRIPTION	8
2. Task "WRITE_DISABLE"	8
5. Task "READ_DATA_BYTES"	8
6. Task "WRITE_DATA_IN"	9
7. Task "WRITE_ID_PAGE_OR_LOCK"	9
8. Task "READ_ID_PAGE_OR_LOCK_STATUS"	9
9. Task "READ_DATA_BYTES_HD"	9

1. PROJECT ARCHITECTURE

a) Project

This project offers a Verilog behavioral model of the SPI Serial EEPROM M95xxx. In order to offer an example of a complete Verilog project, some other Verilog files are offered.

- a) The M95xxx memory Verilog model itself: [M95xxx_Memory.v](#) file,
- b) The set of instructions to transmit to M95xxx memory : [M95xxx_Driver.v](#) file.

These two files are gathered under the [M95xxx_Testbench.v](#) file.

[M95xxx_Macro_Mux.v](#) file is included by all files

[M95xxx_Parameters.v](#) file is included by all files

[M95xxx_Testbench.v](#) initializes memory array with [M95xxx_Initial.dat](#) and [M95xxx_ID_Initial.dat](#) files

[M95xxx_Driver.v](#)

[M95xxx_Memory.v](#)

Decodes SPI protocol
Executes memory read and write operations
Checks received signals AC timing

b) Source files

[M95xxx_Macro_mux.v](#)

It contains all M95xxx memories AC tables. One table is selected with the help of the `+define+` directives.

[M95xxx_Parameters.v](#)

This file defines the parameter values of Memory Size, Number of effective address bits, Page size.

[M95xxx_Testbench.v](#)

Links the [M95xxx_Driver.v](#) to the [M95xxx_Memory.v](#).

[M95xxx_Driver.v](#)

Simulates an SPI bus master transmitting instructions to the [M95xxx_Memory.v](#).

[M95xxx_Memory.v](#)

Describes the behavioral model of M95xxx Memory.

2. HOW TO DEFINE YOUR OWN PROJECT

a) Files to compile

When simulating the project, the following files must be compiled.

1. M95xxx Parameter file ([M95xxx_Parameters.v](#))

This file defines the Memory, that is: Size, Number of effective address bits, Page size. These parameters are defined by a `+define+` directive. During simulation, this file must be included in all other files.

2. M95xxx Macro file ([M95xxx_Macro.v](#))

The user has to define the M95xxx memory to choose within the [M95xxx_Macro.v](#) Macro file. Choosing the memory to simulate is accomplished with the help of *define directives* (see paragraph b) below), the selected Macro file will then contain only the AC characteristics corresponding to the defined memory.

[M95xxx_Macro.v](#) must include the [M95xxx_Parameters.v](#) file with its complete directory path (ex: home/user/simulation_file/M95xxx_Macro.v).

3. [M95xxx_Testbench.v](#)

[M95xxx_Testbench.v](#) must include the [M95xxx_Parameters.v](#) file with its complete directory path (ex: home/user/simulation_file/M95xxx_Macro.v).

4. [M95xxx_Driver.v](#)

This file is an example and can be replaced with some other file (specific to the end application to simulate).

[M95xxx_Driver.v](#) must include the [M95xxx_Parameters.v](#) file with its complete directory path (ex: home/user/simulation_file/M95xxx_Macro.v).

5. [M95xxx_Memory.v](#)

[M95xxx_Memory.v](#) must include the [M95xxx_Parameters.v](#) file with its complete directory path (ex: home/user/simulation_file/M95xxx_Macro.v).

b) How to use the define directives

The Verilog compile provides the following directive variables to define the desired M95xxx product from the *M95xxx_Macro_mux* file.

1. SPI memory density

This parameter must be defined, with the directive “*+define+MXXb*”. The memory density is selected among {*M1Kb*, *M2Kb*, *M4Kb*, *M8Kb*, *M16Kb*, *M32Kb*, *M64Kb*, *M128Kb*, *M256Kb*, *M512Kb*, *M1Mb*, *M2Mb*}

If the device offers the Identification Page

Note: the identification page is offered only by the M95xxx-Dx standard products, please refer to the M95xxx-Dx data sheets for more.

- If the SPI device offers the Identification Page, this parameter has to be defined with the directive “*+define+ID*”.
- If the SPI device does not offer the Identification Page, there no need to define this parameter.

2. Automotive device

Note that the Verilog models support only the automotive M95xxx-A125/145 devices and do not support the automotive M95xxx-125 devices (the M95xxx-125 devices are not recommended for new design)

- If the device is referenced in the data sheet as M95xxx-A125 or M95xxx-A145, directive “*+define+A125/A145*” has to be chosen.

3. *+define+Vcc_range*

This parameter must be defined; it defines the supply voltage range {*W*, *R*, *F*} defined in the M95xxx data sheet.

- If the device reference in data sheet is M95xxx-W, this parameter has to be defined with the directive “*+define+W*”
- If the device reference in data sheet is M95xxx-R, this parameter has to be defined with the directive “*+define+R*”
- If the device reference in data sheet is M95xxx-F, this parameter has to be defined with the directive “*+define+F*”

4. *+define+voltage*

This parameter must be defined; it defines the supply voltage value delivered in the end application.

- If supply voltage value delivered in the end application in within [1.7V-1.8V] range, this must be defined with the directive “*+define+voltage=1.7*”
- If supply voltage value delivered in the end application in within [1.8V-2.5V] range, this must be defined with the directive “*+define+voltage=1.8*”
- If supply voltage value delivered in the end application in within [2.5V-5.5V] range, this must be defined with the directive “*+define+voltage=2.5*”
- If supply voltage value delivered in the end application in within [4.5V-5.5V] range, this must be defined with the directive “*+define+voltage=4.5*”

Example

The following will compile the Verilog model for a M95512-DR (512 Kb serial SPI bus EEPROM, 1.8V/5.5V, with Identification Page, for a supply voltage higher than 2.5V):

```
vlog +define+M512Kb +define+ID +define+R +define+voltage=2.5  
/directory_name/M95xxx_Memory.v /directory_name/M95xxx_Macro.v  
/directory_name/M95xxx_Parameters.v /directory_name/M95xxx_Driver.v  
/directory_name/M95xxx_Testbench.v
```

Note: “directory name” is the directory path and the file name of your files, you should therefore replace “directory name” with your own path name and file name.

c) How to use the initialization file

Initialization files (*M95xxx_Initial.dat* and *M95XXX_ID_Initial.dat*) are offered to define the initial content of the memory array and the memory identification page (only for specific “-D” devices) before starting a simulation. When starting a simulation, the simulated memory array and memory identification page are automatically loaded with the contents of the *M95xxx_Initial.dat* and *M95XXX_ID_Initial.dat* files respectively.

The *M95xxx_Initial.dat* and *M95XXX_ID_Initial.dat* files provided with this Verilog model is filled with “FF” but they can be easily replaced by another user’s initialization files. In order to change the files, the user must change the parameter of system task \$readmemh named *M95xxx_Initial.dat* in the *M95xxx_Testbench.v* file.

Initial file format:

The format used to define 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>

d) Messages when running a simulation

When running a simulation, the M95xxx Verilog model might send messages to prompt the status of the model, as commonly used in most Verilog simulators:

Note:

A note message is normal information.

Warning:

A warning message informs the user that the M95xxx Verilog model is not properly driven (through the SPI bus) and that the SPI sequence is not compliant with the M95xxx specification.

Error:

An error message also informs the user that the M95xxx Verilog model is not properly driven (through the SPI bus) and that the SPI sequence is not compliant with the M95xxx specification.

3. DRIVER FILES DESCRIPTION

Background

- [M95xxx_Driver.v](#) provides an example for driving the M95xxx memory model.
- It is easy to replace [M95xxx_Driver.v](#) with your own driver file by simply changing the [M95xxx_Testbench.v](#) to be linked to [your_new_driver.v](#) file and to [M95xxx_memory.v](#).

[M95xxx_Driver.v](#) file defines seven tasks, every task corresponds to an instruction defined in M95xxx datasheet. In [M95xxx_Driver.v](#), these tasks are invoked to generate driver for memory access.

1. Task “ WRITE_ENABLE ”

This task generates Write Enable instruction sequence, sets WEL bit.

2. Task “ WRITE_DISABLE ”

This task generates Write Disable instruction sequence, resets WEL bit.

3. Task “ READ_STATUS_REGISTER ”

This task generates Read Status Register instruction sequence, reads out the data of Status Register.

4. Task “ WRITE_STATUS_REGISTER ”

This task generates Write Status Register instruction sequence, new value is written to the Status Register.

Format: WRITE_STATUS_REGISTER (sr_data [7:0])

- sr_data [7:0]: the new value that will be written to status register.

5. Task “ READ_DATA_BYTES ”

This task generates Read Data Bytes instruction sequence, read out “n” bytes data from the appointed address.

Format: READ_DATA_BYTES (n, address [address_bits-1:0])

- n: the number of data bytes that will be read out.
- address [address_bits-1:0]: the destination address for read data out.

6. Task “WRITE_DATA_IN ”

This task generates Write Data In instruction sequence, writes data in the appointed memory address.

Format: WRITE_DATA_IN (n, data [7:0], address [address_bits-1:0])

- n: the number of data bytes that will be written in.
- data [7:0]: the value of data that will be written in.
- address [address_bits-1:0]: the destination address for write data in.

7. Task “WRITE_ID_PAGE_OR_LOCK”

This task generates Write Data In instruction sequence, writes data in the appointed address of the ID page or writes to the Lock Identification Page.

Format: WRITE_ID_PAGE_OR_LOCK (n, idn_lock ,data [7:0], address [address_bits-1:0])

- n: the number of data bytes that will be written in.
- idn_lock: selects between the ID page and the Lock ID. 0=id page; 1=Lock ID
- data [7:0]: the value of data that will be written in.
- address [address_bits-1:0]: the destination address for write data in.

8. Task “READ_ID_PAGE_OR_LOCK_STATUS”

This task generates Read Data Bytes instruction sequence, read out “n” bytes data from the appointed address in the Identification page or the read of the locked/unlocked status of the Identification page.

Format: READ_ID_PAGE_OR_LOCK_STATUS (n, idn_lock ,address [address_bits-1:0])

- n: the number of data bytes that will be read out.
- idn_lock: selects between the ID page and the Lock ID. 0=id page; 1=Lock ID Status
- address [address_bits-1:0]: the destination address for read data out.

9. Task “ READ_DATA_BYTES_HD ”

This task generates Read Data Bytes instruction sequence, read out “n” bytes data from the appointed address with HOLD Condition test.

Format: READ_DATA_BYTES_HD (n, id, address [address_bits-1:0])

- n: the number of data bytes that will be read out.
- id: select between memory array or memory identification page. id=0 is memory array; id=1 is identification page
- address [address_bits-1:0]: the destination address for read data out.