

### Verilog model for the F70 technology NAND SLC large page memory devices

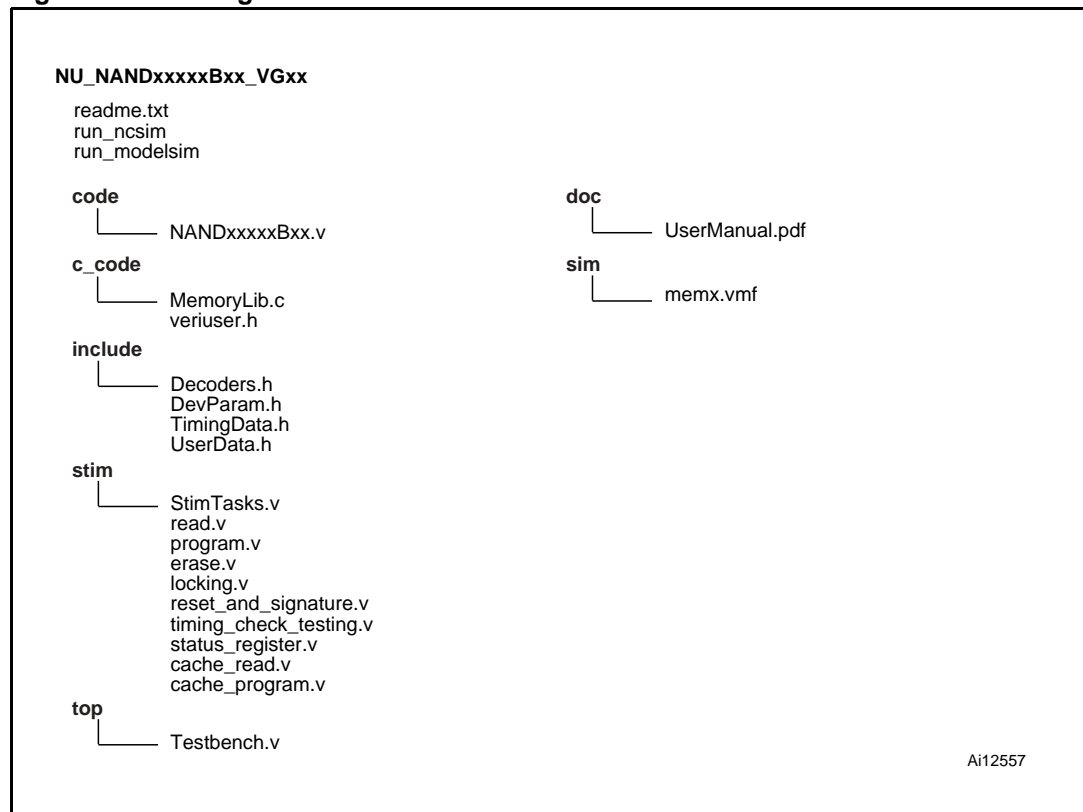
This user manual describes the Verilog behavioral model for NANDxxxxxBxx SLC large page flash memory devices.

## Organization of the Verilog model delivery package

The Verilog model delivery package, *NU\_NANDxxxxxBxx\_VGxx.zip*, is organized into a main directory, named *NU\_NANDxxxxxBxx\_VGxx*, containing seven subdirectories with their related files (see [Figure 1: Package architecture](#)):

1. **code** subdirectory: contains the code source files
2. **c\_code** subdirectory: contains model source files written using C language
3. **include** subdirectory: contains the library source files
4. **doc** subdirectory: contains the model documentation (user manual)
5. **sim** subdirectory: contains the simulation initialization files
6. **stim** subdirectory: contains the stimuli files used for simulation
7. **top** subdirectory: contains the testbench file used for simulation

**Figure 1. Package architecture**



1. See the readme.txt file for the complete list of files contained in each folder.

# Contents

- 1      Device description ..... 3**
  
- 2      Verilog behavioral model ..... 4**
  - 2.1    Modules and libraries ..... 4
    - 2.1.1    Modules ..... 4
    - 2.1.2    Header files ..... 5
    - 2.1.3    C libraries ..... 5
    - 2.1.4    Verilog Testbench and Stimuli files ..... 5
  
- 3      Simulation guidelines ..... 6**
  - 3.1    Launching a simulation ..... 6
  - 3.2    Simulation timings ..... 6
  - 3.3    memory\_file file format ..... 7
  - 3.4    Customize model and simulation ..... 7
  
- 4      Verilog types used in model ports ..... 8**
  
- 5      Revision history ..... 8**

# 1 Device description

NANDxxxxxBxx SLC large page flash memory devices are a family of 2112-byte/1056-word page non-volatile flash memories that uses NAND cell technology. The devices range from 1 Gbit to 4 Gbits and operate with either a 1.8 V or 3 V voltage supply. The size of a page is either 2112 bytes (2048 + 64 spare) or 1056 words (1024 + 32 spare) depending on whether the device has a x8 or x16 bus width.

The address lines are multiplexed with the Data Input/Output signals on a multiplexed x8 or x16 input/output bus. This interface reduces the pin count and makes it possible to migrate to other densities without changing the footprint.

Each block can be programmed and erased over 100,000 cycles. To extend the lifetime of NAND flash devices it is strongly recommended to implement an error correction code (ECC).

The devices feature a write protect pin that allows performing hardware protection against program and erase operations.

The devices feature an open-drain ready/busy output that can be used to identify if the program/erase/read (P/E/R) controller is currently active. The use of an open-drain output allows the ready/busy pins from several memories to be connected to a single pull-up resistor.

A Copy Back Program command is available to optimize the management of defective blocks. When a page program operation fails, the data can be programmed in another page without having to resend the data to be programmed.

Each device has cache program and cache read features which improve the program and read throughputs for large files. During cache programming, the device loads the data in a cache register while the previous data is transferred to the page buffer and programmed into the memory array. During cache reading, the device loads the data in a cache register while the previous data is transferred to the I/O buffers to be read.

All devices have the chip enable don't care feature, which allows code to be directly downloaded by a microcontroller, as Chip Enable transitions during the latency time do not stop the read operation.

All devices have the option of a unique identifier (serial number), which allows each device to be uniquely identified.

The unique identifier options is subject to an NDA (non disclosure agreement) and so not described in the datasheet. For more details of this option contact your nearest Numonyx sales office.

## 2 Verilog behavioral model

The NANDxxxxBxx Verilog behavioral model is contained in the *NANDxxxxBxx.v* file of the **code** subdirectory. It includes a set of modules that implement all device functions listed in the device datasheet.

These modules use a set of parameters defined in specific header files (contained in the **include** subdirectory).

Moreover certain model functions are implemented using C language; in this case, a PLI library is used to transfer data between 'C code' and 'Verilog code'. The section of the model written in C language, is placed in the **c\_code** subdirectory.

*Note:* Please check the Numonyx web site or contact your local Numonyx sales office for the most recent version of the device datasheet.

*Please refer to readme.txt file in the main package directory for reference datasheet used during model development and validation.*

*This model was validated using a Cadence NC-SIM 5.7 simulator. The use of this model with other simulators is not guaranteed.*

### 2.1 Modules and libraries

The NANDxxxxBxx Verilog modules and libraries are described in the following sections.

The **code/NANDxxxxBxx.v** Verilog file and C library code files must be compiled in the same order as specified in the *run\_ncsim* file.

#### 2.1.1 Modules

**NANDxB** is the 'core' of the model: latches signals, data, address and commands, and make use of all others modules.

**UtilFunctions** contains utility functions used in various parts of the model.

**CUIdecoder** decodes command sequences of the flash memory.

The **Memory** module implements the basis operations for read and write memory matrix: update of page index, data transfer between page buffer and flash memory.

**Program** implements program and erase operations.

**Read** models read operations.

The **OutputBusManager** module is used for modeling output bus (output data are written on the bus in accordance with expected timings).

**LockManager** implements features to protect device against program and erase operations (locking features).

**StatusRegister** models the status register of the flash memory.

**Signature** contains the electronic signature data.

During the simulation, the **TimingCheck** module checks the timing of input signals to ensure that related constraints are respected.

### 2.1.2 Header files

The following section describes the header files, used in the model.

The **Decoders.h** file is used in NANDxB module; it contains all the instances of the CUIdecoder module (each instance recognizes a specific command sequence of flash memory).

**UserData.h** contains definitions that can be changed by users: device specification and timingCheck option; this is the only header file that can be modified by the user, and it will be described in the next section, Customize model and simulation.

The **DevParam.h** file contains definitions of constants related with memory characteristics, and used in various parts of the model.

**TimingData.h** contains the definitions of the timing constraints.

### 2.1.3 C libraries

In order to optimize simulation performance, memory array data structure is implemented in C language as a 'dynamic list of pages' (each node of the list contains an array which represents one page; only the pages programmed by the user, are present in the list).

The **MemoryLib.c** file contains definition of the dynamic list representing the memory array, and definition of tasks which operates on this data structure.

These C tasks are used as system tasks in the Verilog code.

To transfer data between 'C code' and 'Verilog code', the PLI library is used.

PLI is a standard library and is defined in the **veriusers.h** file in the **c\_code** directory.

### 2.1.4 Verilog Testbench and Stimuli files

The **top** subdirectory of the Verilog model delivery package contains a testbench file, *Testbench.v*, used to simulate the model using the various stimuli files.

Stimuli files are available in the **stim** subdirectory. They cover many operational conditions of the device, and in particular, the Command User Interface (CUI) commands.

The testbench and the stimuli files are written using the standard Verilog language and make use of specific Verilog tasks contained in the **stim/StimTasks.v** file

## 3 Simulation guidelines

### 3.1 Launching a simulation

*run\_ncsim* is an example of script used to launch the Cadence NC-SIM simulation. It is located in the main directory. This file compiles and elaborates the Verilog model file and the stimuli files contained into the **stim** directory.

To build the simulation script (for Cadence NCSIM, or Mentor Modelsim simulators), the following steps must be respected:

1. Compile C code and build dynamic library

[Linux and Solaris / all simulators]:

```
gcc -c <C_code_file_name.c> -o <object_file_name.o>
ld -G <object_file_name.o> -o <dynamic_library_name.so>
```

2. Compile Verilog code

[NCSIM]:

```
ncvlog -cdslib <cds.lib path> -hdlvar <hdl.var path>
<file_to_be_compiled.v>
```

[Modelsim]:

```
vlog -work <work_dir path> <file_to_be_compiled.v>
```

3. Elab (only for NCSIM)

```
ncelab -cdslib <cds.lib path> -hdlvar <hdl.var path> -
loadpli1 <dynamic_library_path>:bootstrap_fun
<snapshot_name>
```

4. Launch simulation

[NCSIM]:

```
ncsim -cdslib <cds.lib path> -hdlvar <hdl.var path>
<snapshot_name> -gui
```

[Modelsim]:

```
vsim <top_level_module> -pli <dynamic_library_path>
```

### 3.2 Simulation timings

To reduce the simulation time, the user can reduce certain program and erase times in the Verilog simulation model. These values can be configured by setting the following variables defined in the *TimingData.h* library file:

- read\_delay
- program\_delay
- erase\_delay
- cacheProg\_delay
- exitCacheRead\_delay
- busy\_delay

For instance, if the user wants change the *program\_delay* time to 100 ns, he can redefine *program\_delay* constant as follows:

```
parameter program_delay = 100;
```

### 3.3 *memory\_file* file format

To facilitate testing of the memory model, the memory array can be loaded with specific data at power-up.

The format of the *memory\_file.vmf*, located in the **sim** subdirectory, must be as follows:

```
@hex_address
hex_data
hex_data_1
.....
```

where *hex\_data* is stored at location *hex\_address*, *hex\_data\_1* at the location *hex\_address + 1*, and so on ...

As an example:

```
@07FFFF
C14B
129A
.....
```

Comments in memory file lines are allowed using the notation:  `/ / comment`

The model is delivered with a template memory file in the **sim** directory called *memx.vmf*.

The name of memory file is defined as parameter of NANDxB module. It can be specified in the stimuli file, using the following syntax:

```
defparam testbench.DUT.memory_file = "memory_file_name";
```

If the user does not provide the initialization file (`memory_file:= " "`), all the memory bits are loaded with '1', therefore the entire array is erased.

### 3.4 Customize model and simulation

Certain features and characteristics of the model and the simulation process are customizable by the user. The parameters that the user can modify are contained in the **UserData.h** header file.

The parameters whose value can be changed are:

- Device name definition  
This parameter is the first definition contained in **UserData.h**, and specifies which device is described by the model (each device is characterized by memory size, bus width, supply voltage and timing constraints).
- TimingChecks  
If this string is set to 'on' value, all timing checks will be performed during the simulation. Otherwise, if it is set to 'off', no timing checks are performed.

## 4 Verilog types used in model ports

The port section of NANDxxxxxBxx Verilog model defines the name and the related type for each signal of the device, as shown in [Table 1](#).

**Table 1. Model ports**

Port	Type	Description
Vdd	[31: 0] Input wire <sup>(1)</sup>	Supply voltage
PRL <sup>(2)</sup>	Input wire	Power-up read enable, lock/unlock Enable
WP_N	Input wire	Write protect
AL	Input wire	Address latch enable
CL	Input wire	Command latch enable
E_N	Input wire	Chip enable
R_N	Input wire	Read enable
W_N	Input wire	Write enable
IO	[busDim-1: 0] Output wire <sup>(3)</sup>	Input output bus
RB_N	Output wire	Ready / busy signal
dump <sup>(4)</sup>	Input wire	Dump memory

1. Voltage signal is represented with 32-bit binary array, which decimal value corresponds to voltage value in millivolts.
2. This input is present only for the devices with PRL input signal.
3. busDim = 8 or 16, depending on whether device bus width is 8 or 16 bits.
4. Additional input of the model (not present in the real device); is used for dumping memory content in a file; in current version of the model dump feature is not yet implemented, then this input must be left not connected (high impedance).

## 5 Revision history

**Table 2. Document revision history**

Date	Revision	Changes
17-Apr-2007	1	Initial release.
11-Aug-2008	2	Modified document's title and applied Numonyx branding.

**Please Read Carefully:**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH NUMONYX™ PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN NUMONYX'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, NUMONYX ASSUMES NO LIABILITY WHATSOEVER, AND NUMONYX DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF NUMONYX PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Numonyx products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Numonyx may make changes to specifications and product descriptions at any time, without notice.

Numonyx, B.V. may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Numonyx reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Numonyx sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Numonyx literature may be obtained by visiting Numonyx's website at <http://www.numonyx.com>.

Numonyx StrataFlash is a trademark or registered trademark of Numonyx or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 11/5/7, Numonyx, B.V., All Rights Reserved.