

Portland State University
ECE 593 winter 2017
Fundamentals of Presilicon Functional Validation

*Verification of NAND Flash Memory Controller
Design*

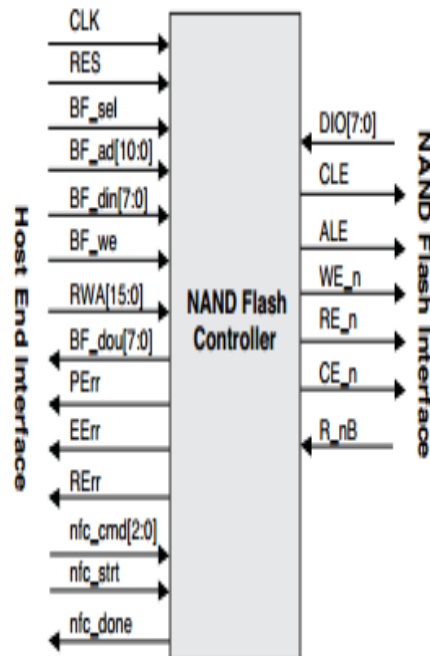
Verification technocrats:

Vinod Sake
Rajesh Ceervi
Lokesh Astakar

Course Instructor

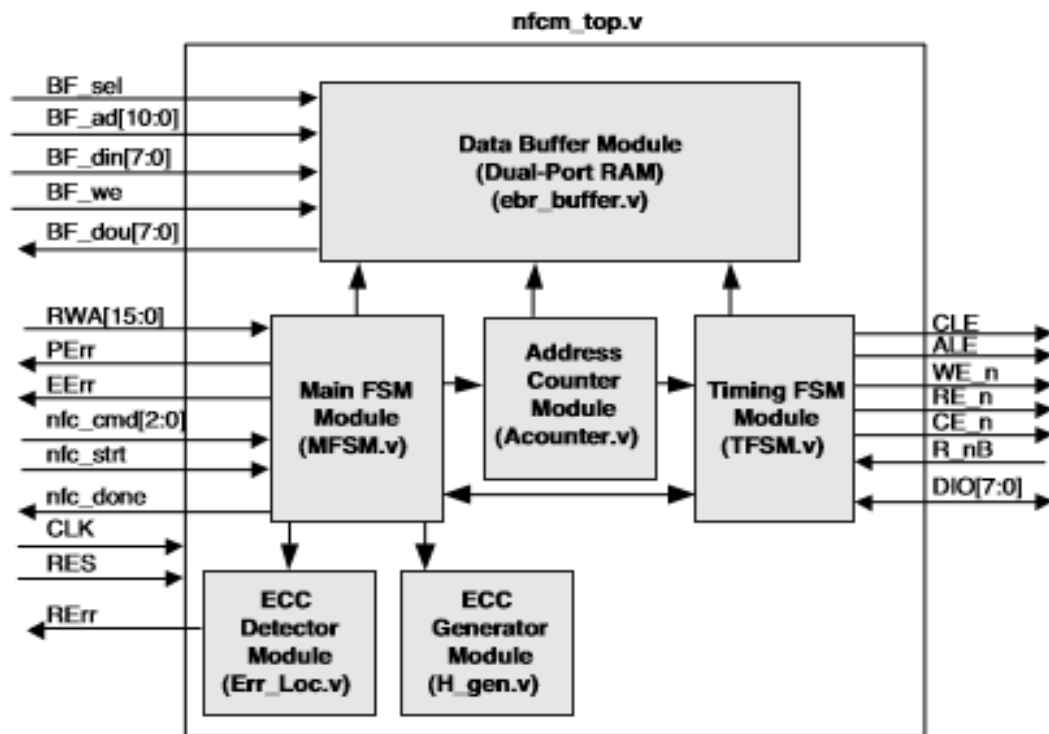
Prof. Tom Schubert

Lattice Semiconductor NAND FLASH MEMORY CONTROLLER Design and Specification:



Signal Name	Signal Direction	Active State	Definition
Host End interface			
CLK	Input	N/A	Clock signal.
RES	Input	High	Reset signal.
BF_sel	Input	High	Dual-port RAM clock enable signal.
BF_ad[10:0]	Input	N/A	Dual-port RAM address signal.
BF_din[7:0]	Input	N/A	These lines are used to pass data to the dual-port RAM.
BF_we	Input	High	Dual-port RAM write signal.
RWA[15:0]	Input	N/A	These address signals are used by the Flash device.
BF_dout[7:0]	Output	N/A	These lines are used to pass data to host.
PErr	Output	High	Page program operation error signal.
EErr	Output	High	Block erase operation error signal.
RErr	Output	High	Page read operation error signal.
nfc_cmd[2:0]	Input	N/A	Command code signal.
nfc_strt	Input	High	When asserted, indicates the host initiates a operation.
nfc_done	Output	High	When asserted, indicates an operation is done.
NAND Flash Interface			
DIO[7:0]	In/Out	N/A	I/O pins used to send commands, address, and data to the Flash, and receive data during read operations.
CLE	Output	High	Command Latch Enable.
ALE	Output	High	Address Latch Enable.
WE_n	Output	Low	Write Enable.
RE_n	Output	Low	Read Enable.
CE_n	Output	Low	Chip Enable.
R_nB	Input	N/A	When this signal is high, the Flash is ready for the next operation. When it is low, an internal operation is in progress.

NAND flash memory controller RTL Implementation:



Reference Design supports following operation

- Reset
- Read ID
- Erase (Per Block Bases)
- Program page(copy content of data buffer into Flash Memory)
- Read page(content of Flash page is copied into the data buffer)
- Read Status

Module Description:

Data Buffer Module—This Module is implemented as Dual Port RAM which facilitates storing of data when host writes data into the Flash and Reads data from the Flash.

Main FSM-This FSM works together with timing FSM where the module interprets command from the host and passes control signals to the timing FSM which creates all the necessary controls for NAND flash to execute the repeated task with strict timing requirements.

Address Counter Module-Generates Address control signals required for the data buffer module based state machine in the main FSM.

ECC Generator Module-Generates Error correction code during program operation and stores the ECC code in Flash memory.

ECC Detector Module- Makes use of this ECC code in Flash memory to detect errors in the data during host read operation.

Design Exercise/Verification Plan:

Verification of Nand Flash Memory controller at System Level as Lattice Semiconductor Company has provided functional specification only for the top level model.

Verification of the design using Basic Functional Test cases:

- Verify the basic command response protocol at the command port.
- Verification of basic operation of all the command on the command port like Read, Page Erase, Write, Reset(Read and write abort), Read ID and Read
- Check if the valid cmd, address and data has been pumped out from the Nand memory controller to flash memory for the corresponding command from the host.

Verification of the design using Advanced Functional Test cases:

- Check for the design behavior with system reset and read/write command operation applied simultaneously.
- Check for the design behavior with a read followed by write operation with the same address.
- Check for the design behavior with a write followed by read with the same address.
- Check for 5 successive read operation with same address.
- Check for 5 successive read operation with different address.
- Check for 3 successive write operation with same address.
- Check for 3 successive write operation with different address.
- Check for read and write operation with invalid address.
- Check for read ID operation if it is reading correct ID of the device.
- Check for the read/write operation followed by erase operation.
- Check for the response of the design by driving command port without driving the start signal.
- Check if the design raises any error flag if any erroneous data is received while read operation. (Raise a flag).
- Check for the signal protocol from the memory controller:

	RE_n	WE_n	ALE	CLE	CE_n
Command	high	low	low	high	low
Address	high	low	high	Low	Low
I/O data	Low(only Read)	Low(write)	low	low	low

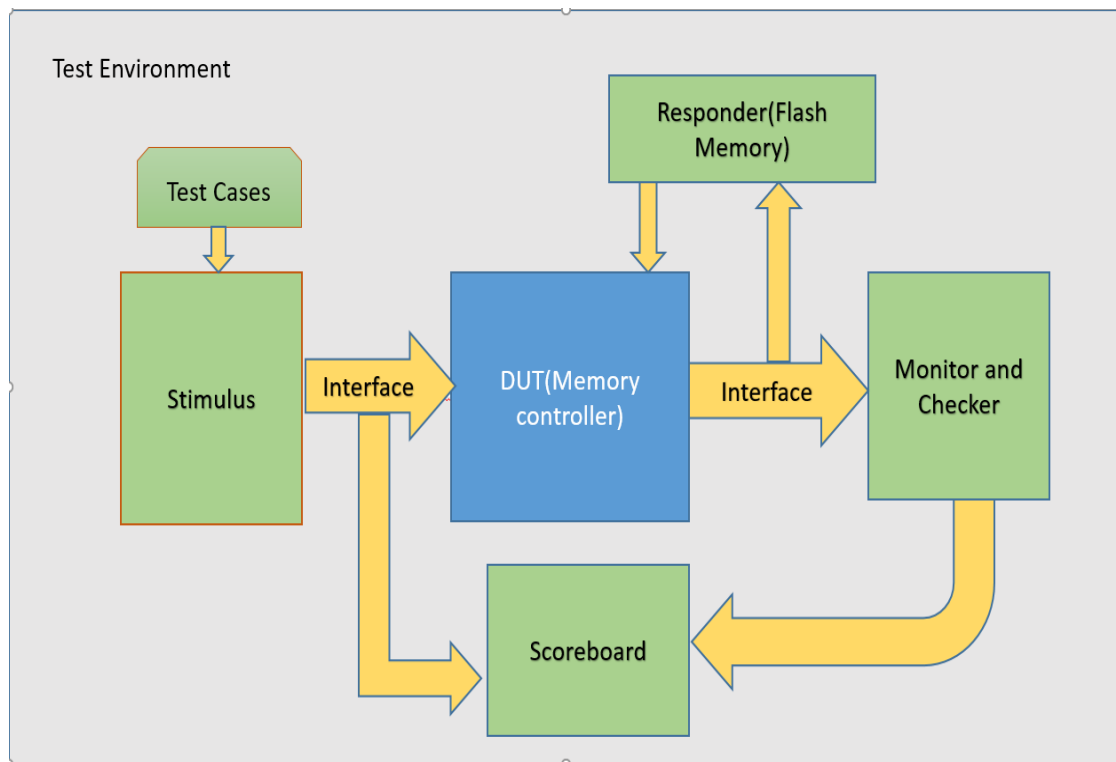
- Check for the nfc_done signal when the intended operation is done from memory stub.
- Check for the valid address and command generated by the memory controller as per the specification.

Verification of the Generic Functional Tests and check:

- Check if the design can handle invalid command.
- Check if the design under test generates superfluous cmd, address or data values.

- Check if system reset functionality works correctly.

Verification Strategy and Test bench environment:



- Implemented Oups based verification test bench to create modularity and reusability of the verification of the environment.
- Implemented Bus Functional Model to interface all the signals with the stimulus, Scoreboard and Checker test bench components.

Stimulus:

- Stressed the design under test with basic, generic and advanced test cases as described earlier.
- Initially, Applied basic **deterministic test cases** to check if the design responds correctly to the all the basic operations as per the functional specification.
- **Advanced deterministic test cases** were driven to stress the design completely.
- Later, **Random stimulus** was generated to check the design for **any un-usual behavior** and to determine if there was any design space that was left unattended. Command, data and address are also randomized.

Scoreboard:

- Scoreboard was designed to contain a reference model of the design which provides the expected address and command for the corresponding request issued by the host interface device.
- Scoreboard employs Queue/Memory to hold the commands issued back to back from the host interface. Scoreboard also has the ability to invalidate the successive commands that are issued by the Host after determining the valid command.
- Scoreboard sends Checker with an expected results that needs to be checked by the checker when a particular signals are triggered at the output.
- Reference model is implemented using FSM.

On the Fly Checker:

- Based on the design, we decided to implement on the fly checking.
- Particular signals are monitored at the checker and any changes in the signal seen by the checker at the DUT output calls the scoreboard to check if there are any protocols the needs to be checked.
- Checker also performs protocol checking of the command issue from the host.

Memory Stub/Responder:

- Leveraged the design of memory stub provided by Lattice semiconductor to include 32 blocks, where each block contains 64 pages and each page can store (2K + 64) bytes of data as opposed to 2K memory that was provided initially. The design memory can be parameterized to any memory size
- This extension of memory would enable us to perform exhaustive verification on the successive commands issued from the host. I.e. successive reads and writes to same page and different pages of the block.

Coverage:

- All ports are exercised on the design.
- Different commands are exercised on single port.
- Sequence of the commands exercised on the design.
- Multiple ports are simultaneously stressed.

Exit Criteria and Time Spent:

- 2 days to understand the specification
- 6 hours for overall verification plan
- 6 days to develop initial functional test bench environment.
- 3 hours to fix the bugs in the verification environment.
- 3 hours for running basic deterministic test cases and seeing the behavior of the design.
- 4 hours of running advanced deterministic test cases and observing the behavior of the design.

- 6 hours to run the random generated stimulus and checking for 100 of random test cases.
- 9 hours to improve design of test bench environment to include coverage groups and cross coverages.
- Successfully completed verifying the design as per the proposal on March 20th.

RTL Design Roadblocks:

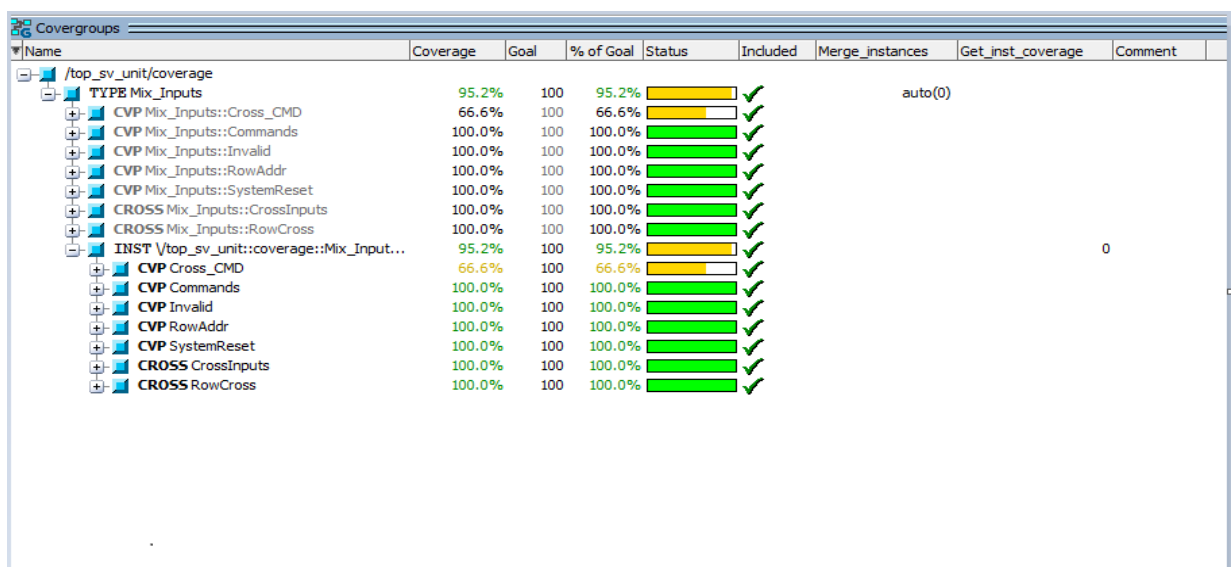
- Design included data buffer that could be used only with the licensed Lattice semiconductor tool.
- Carefully studied the RTL design and developed data buffer from scratch that functions as per the design.
- Specification of only top level module provided by the company hence forced us to do black box testing.

Interesting Scenarios tested and result observed:

- Multiple writes to the same address— **works perfectly well.**
- First read without writing anything to the memory- **Reads the data that was initialized with.**
- Issuing reset command in between read and write command to observe if the read or write operation is halted--- **Read and write just refuses to get halted**
- Erasing particular block of the memory- **Erase mechanism not include in the stub, this will be our future work.**

Results Achieved:

- Achieved 96% coverage on the design under test which indicates that most of our test cases has been exercised.
- Verified the design completely and found the design to be bug free and the design is now ready for tape out😊



Name	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/top_sv_unit/coverage	95.2%	100	95.2%					
TYPE Mix_Inputs	95.2%	100	95.2%					
CVP Mix_Inputs::Cross_CMD	66.6%	100	66.6%					auto(0)
CVP Mix_Inputs::Commands	100.0%	100	100.0%					
CVP Mix_Inputs::Invalid	100.0%	100	100.0%					
CVP Mix_Inputs::RowAddr	100.0%	100	100.0%					
CVP Mix_Inputs::SystemReset	100.0%	100	100.0%					
CROSS Mix_Inputs::CrossInputs	100.0%	100	100.0%					
CROSS Mix_Inputs::RowCross	100.0%	100	100.0%					
INST \top_sv_unit::coverage::Mix_Input...	95.2%	100	95.2%					0
CVP Cross_CMD	66.6%	100	66.6%					
CVP Commands	100.0%	100	100.0%					
CVP Invalid	100.0%	100	100.0%					
CVP RowAddr	100.0%	100	100.0%					
CVP SystemReset	100.0%	100	100.0%					
CROSS CrossInputs	100.0%	100	100.0%					
CROSS RowCross	100.0%	100	100.0%					

Problems Encountered:

- Challenges faced while making the test bench component communicate with each other.
- Reverse Engineered the RTL design further to get insights into the functional specification.
- Functional specification did not mention clearly on how many cycles does each operation take to finish it and it took most of our time to figure out time cycles for each operation which would be required for checking.

List of Bugs found and injected bugs:

- Reset command should halt any read or write operation but we found that this was not happening as per the specification.
- There was not much bugs that was found as this is commercial product to check if our checker is checking things correctly.
- We injected few bugs in the design and we changed the command and address in the FSM as opposed to specification in the design and our checker threw errors as we expected.

Environmental Setup:

- All the test bench components was developed in System Verilog.
- These test bench are strictly developed to run on QuestaSim EDA tool.

Ideas Leveraged:

- Much of the ideas was obtained by studying TinyALU test bench and referring websites like verification Academy.
- Open ended discussion with Professor Tom Schubert gave us more insight into test cases that the design could be tested.

Next steps:

- Upgrade the verification environment in UVM which can be completed by end of June.
- Implement Erase functionality in the memory stub.
- Expand the memory to include more blocks to perform exhaustive verification.

References:

- RTL designed picked up from Lattice Semiconductor official site.
- Prof Tom Schubert's Pre-Silicon verification slides.