TLM modeling and simulation for NAND Flash and Solid State Drive systems

Tim Kogel, Victor Reyes - Synopsys
Agenda

• Introduction to NAND Flash storage
• Solid State Drive challenges
• Virtual Prototyping
• SSD Reference VDK overview
• Summary
Introduction

• Growing demand for Solid State Drives in Enterprise and Client markets
  – Time to market is shortening
  – NAND Flash is the technology of choice

• SSD is non-volatile storage that can be electrically erased and reprogrammed
  + High durability (vs. hard disks)
  + Fast access times (similar to DRAM)
  – Finite number of writes to a block after which it wears out

Source: Micron
ONFI protocol

• Open NAND Flash Interface Specification
  – [www.onfi.org](http://www.onfi.org)

• 34 standardized ONFI commands (composed of cycles)
  – 1 or 2 command cycles
  – 0, 1, 3 or 5 address cycles
  – Variable data read/write cycles

<table>
<thead>
<tr>
<th>Mandatory Commands</th>
<th>Optional Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Multi-plane</td>
</tr>
<tr>
<td>Change Read Column</td>
<td>Copyback Read</td>
</tr>
<tr>
<td>Block Erase</td>
<td>Change Read Column</td>
</tr>
<tr>
<td>Read Status</td>
<td>Read Cache Random</td>
</tr>
<tr>
<td>Page Program</td>
<td>Read Cache Sequential</td>
</tr>
<tr>
<td>Change Write Column</td>
<td>Read Cache End</td>
</tr>
<tr>
<td>Read ID</td>
<td>Multi-plane</td>
</tr>
<tr>
<td>Read Parameter Page</td>
<td>Read Status Enhanced</td>
</tr>
<tr>
<td>Reset</td>
<td>Multi-plane</td>
</tr>
<tr>
<td>Copyback Program</td>
<td>Page Cache Program</td>
</tr>
<tr>
<td>Multi-plane</td>
<td>Synchronous Reset</td>
</tr>
<tr>
<td>Small Data Move</td>
<td></td>
</tr>
</tbody>
</table>
SSD CHALLENGES
NAND flash limitations

• Block erasure
  – Erasure is always at a block level

• Memory wear
  – NAND flash memory wears out, typically after 100k to 1M program/erase cycles
  – Wear leveling balances write operations among blocks to avoid loss of capacity

• Read disturb
  – Frequent reads between erases may change content of nearby cells
  – The block’s content must copied over to another location and original block erased
System complexity

• Performance
  – Dominated by controller latency and memory interface speeds
  – Parallel NAND operations are required to scale bandwidth and hide latencies
  – Fast host interfaces such PCIe (NVMe) are required

• Reliability
  – Improved through ECC and overprovisioning

• Security
  – Encryption
Firmware complexity

- SSD controllers are embedded processors executing firmware-level software functions
  - Bad block mapping
  - Read and write caching
  - Encryption
  - Error detection and correction via Error-correcting code (ECC)
  - Garbage collection
  - Read scrubbing and read disturb management
  - Wear leveling

- Flexible to support evolving algorithms and interface standards

- Optimized to provide highest performance
VIRTUAL PROTOTYPING
Why do Virtual Prototyping?

- Break Dependencies on RTL Availability (by using Transaction Level Models)
- Agile Software Development in Lock Step with Virtual Prototype Development
Virtualizer Development Kits (VDKs)

- Software Development Kits that use a Virtual Prototype as a target
- VDK’s are fully functional models of the system executing target code (SW / FW)

Early Availability
Easier Deployment
Better SW Development Productivity
  - Visibility
  - Control and repeatability
  - Fault Injection support
  - Scriptable
Benefits and opportunities for SSD

• Mitigate risk
  – Try new software, architectures and components before starting implementation

• Early firmware development
  – Develop, integrate and test controller firmware before silicon

• Customer enablement
  – Share VDK based Software development platform with device maker

• Improve reliability
  – Use fault injection capabilities to ensure robustness of firmware
SSD REFERENCE VDK OVERVIEW
Methodology reference kits

**SSD Reference VDK**

- **Bare metal testing SW**
  - For integration test build and source

- **SSD firmware**
  - FTL, NAND, PCIe driver,…

**Initial Design**

- ARM Cluster, memory, …

**Custom VDK**

- **Integr. Tests**
  - refined

- **Custom FW**
  - refined

- **Custom Design**
  - refined

**SSD Reference VDK**

- Jump start your custom VDK Development
- Simple Customization of your ARM CPU subsystem
- Extend with your own IP or Synopsys IP Libraries
- Simple Firmware booting out of the box as a reference
- ONFI compliant TLM model examples

**SSD Reference VDK**

- SSD firmware
- Bare metal testing SW
- Initial Design
- Custom VDK
- SSD Reference VDK

© Accellera Systems Initiative
SSD Reference VDK Design

• Virtualizer Studio based VDK
• Easy reconfiguration of CPU subsystem
  – Any number and types of cores
  – Automatic connectivity
• Largest library of TLM models for interface IP
  – DesignWare PCIe, USB3, SATA
• ONFI compliant configurable TLM models
  – NAND Flash controller, including software driver
  – NAND Flash memories with configurable number of LUN, planes, blocks, pages, etc.
• Script based FTL command generation with Python or TCL
SSD Reference VDK in Virtualizer Studio
NAND Flash Modeling Approach

• Separate Controller from Target devices
  – Explicitly model the NAND Flash interface at the TLM level

• Provide reusable building blocks that encapsulate the NAND Flash protocol details
  – Protocol engine(s) can be share across models
  – Verify once, reuse many times
  – Include best practices for simulation speed using callbacks and analysis instrumentation

• Include solution for model unit-testing
  – Based on reusable objects and TLM Creator unit testing framework

• File based implementation to deal with high density storage
  – Encapsulated as a reusable ‘Logical Unit’ object with debug and analysis instrumentation
  – Highly configurable

© Accellera Systems Initiative 2017
ONFI protocol TLM Modeling

Protocol engine object allows different levels of timing and accuracy on the interface:

- Single transfer with overall time (fastest)
- Separate transfers for command, address and data (more accurate)
Unit-Testing infrastructure

- Automated execution and reporting
- SystemC aware assertions

```c
void write_TheRegister() {
  P.write32(TheRegister, 0x01);
  SCML2_ASSERT_THAT(P.read32(PERIOD.equals(0));
}
```

```c
void write_TheRegister() {
  P.write32(TheRegister, 0x01);
  SCML2_ASSERT_THAT(P.read32(PERIOD.equals(0));
}
```

```c
void write_TheRegister() {
  P.write32(TheRegister, 0x01);
  SCML2_ASSERT_THAT(P.read32(PERIOD.equals(0));
}
```
ONFI Unit-Testing example

The test example below:
1. configures the target proxy
2. programs the controller to execute a read page command
3. waits until the command is completed
4. asserts that the command received is the one programmed

```c
void test_read_page_operation() {
    ...
    target_proxy.set_chip_enable(0);
    target_proxy.set_ready();
    target_proxy.set_data_to_read(tmpData, len);
    this->apb.write32(AC_FIFO, (B_CT_COMMAND|B_WFR | B_CE_0|  tlm_onfi::READ));
    this->apb.write32(AC_FIFO, (B_CT_ADDRESS|column & 0xFF));
    this->apb.write32(AC_FIFO, (B_CT_ADDRESS|(column >> 8) & 0xFF));
    this->apb.write32(AC_FIFO, (B_CT_ADDRESS|rowAddr & 0xFF));
    this->apb.write32(AC_FIFO, (B_CT_ADDRESS|(rowAddr >> 8) & 0xFF));
    this->apb.write32(AC_FIFO, (B_CT_COMMAND|B_WFR|B_CE_0|tlm_onfi::READ_PAGE));
    sc_core::wait(target_proxy.get_end_command_event());
    SCML2_ASSERT_TRUE(target_proxy.received_command_is(tlm_onfi::CMD_READ_PAGE));
}```
SSD Reference VDK Capabilities

- Execute, debug and test full SSD software stacks
  - Debug with full visibility and correlation on software and hardware events
  - Automated testing for a large number of scenarios
  - Analyze software statistics and utilization
- Analyze ONFI metrics
  - Utilization of the NAND Flash controller
  - Number and type of ONFI commands per memory
  - Per block, number of erase operation
  - Per page, number of read and program operation
- Configure and inject errors
  - Factory defect mapping and dynamic data corruption
SSD Reference VDK Software

- SW based on the “Cosmos OpenSSD platform”
  - http://www.openssd-project.org
- Page-level mapping
  - Emulates the functionality of an HDD
  - Every logical page is mapped to a physical page
  - Redirect each write request from the host to an empty area already erased
- Garbage collection
  - Reclaim new free blocks for future write requests
  - Greedy algorithm: victim block is selected to minimize search time
- NAND Controller driver
  - Provides basic commands: block erase, read page, program page, read status, etc.
  - Provide data corruption fault checking
Hardware / Software analysis

Scenario: NAND Flash Controller driver performing a reset command
ONFI Statistics

- ONFI bus utilization in percentage (Active / Idle)
- IOPS (cycles per seconds)
- Number and type of ONFI commands per target
- Number of erased blocks
- Number of read and program operations
- Number of read and program operations (sliced by page)
Factory Defects and Error injection

SSD Reference VDK

Factory defect mapping (scriptable)  Inject data corruption (scriptable)

```python
def set_factory_settings(self):
    print("Initializing Factory Settings ...")
    for i in range(len(self.Nand)):
        self.Nand[i].send_command('set_BadBlock 0')
        self.Nand[i].send_command('set_FactoryInvalidBlock 0 50')
        self.Nand[i].send_command('set_FactoryInvalidBlock 0 367')
        self.Nand[i].send_command('set_FactoryInvalidBlock 0 1112')
        self.Nand[i].send_command('set_FactoryInvalidBlock 0 4095')
```

```python
def inject_data_corruption(self, nandIndex):
    print("Corrupting data in next accessed page ...")
    self.Nand[nandIndex].send_command('set_DataCorruption 10')
```

Configurable number of corrupted bits
Error injection analysis

SSD Reference VDK

Software routines check and clear the ECC fields (assuming data integrity is recovered by ECC hardware).

Error is exposed to the Software through the controller registers (ECC_ERROR_LAST).

Next Read Page operation triggers ECC error (detected by the NAND Flash Controller).

Inject data corruption on NAND Target (scripting).
SUMMARY
Summary

• SSD market is growing rapidly

• Use Virtual Prototyping to manage SSD development risks and challenges
  – Software complexity  → start firmware development months before hardware
  – Time to market  → enable customers early across the supply chain
  – Performance optimization  → analyze hardware/software interactions and trend
  – Data reliability  → validate many (error) scenarios with regression testing

• The SSD Reference VDK is the best starting point to engage!
  – Ready out of the box with configurable models and example software
  – Easy to customize and match an specific customer design
  – Great analysis and scripting capabilities
References

• “Overview of the NAND Flash High-Speed Interfacing and Controller Architecture”, Nina Mitiukhina, IEE 5008 Memory Systems Final Report 0060805

• “Open NAND Flash Interface Specification, revision 4.0”, 2014, www.onfi.org


• “NAND Flash 101: An Introduction to NAND Flash and How to Design It Into Your Next Product”, TN-29-19, www.micron.com

Thank You!
Questions?

tim.kogel@synopsys.com
victor.reyes@synopsys.com