Verifying an SSD Controller from IP Block to Full SoC

Jason Polychronopoulos
Ben Whitehead

MemCon, Santa Clara
Oct 2016
What is Storage?

- All non-volatile memory devices and enclosure systems.
  - HDD
  - SSD
  - USB Thumb Drives
  - Backup (LTO)
  - Storage Systems
    - NAS
    - SAN
    - SDS
SSD Controller SOC
Verification Considerations and Challenges

How do you Verify this?

SSD Controller SOC
- Host Interface
  - ECC Engine
  - Logical to Physical Translation
  - Encrypt/Decrypt Engine
  - NAND Memory Interface
- NAND Array
  - ONFI
  - DDR3 / DDR4 / DDR...
- RAM
  - DDR3 / DDR4 / DDR...
- Software/Firmware
- DDR Controller
- CPU/RISC Processors
- Defect Management, Garbage Collection
- Write Abort
- Read & Program Disturb
- NAND Memory Interface
- XOR engines

Host Controller Interface

SoC sizes and complexity are increasing

FW, Algorithm & Features
- ECC
- Wear Leveling
- Parity Checks (LDPC)
- DFT

NAND
- Functionality
- Size: Terabytes

Host protocols
- PCIe/NVMe
- SATA
- SAS

Software
- Firmware
- Device Drivers
- Applications

System
- Performance
- Power
- Architecture

© Mentor Graphics Corp. Company Confidential
www.mentor.com
Initial Verification Plan

- Use simulation to achieve coverage
- Block level
  - Thorough code and functional coverage targets
  - Use UVM with constrained random and directed tests
  - Re-use available verification IP
- SoC
  - Connectivity tests
  - Functional coverage targeting interactions between blocks
  - Code coverage to ensure all blocks are exercised
    - But not necessarily all code in a block
UVM enables verification re-use

- Definition and separation of verification environment
  - Test from Test Bench (Env)
  - Config, Register Model, Agents, Interfaces
  - Enables test and component interoperability and re-use

- Industry standard
  - Open source SV base class library
  - Widely adopted and supported
  - Enables tools and engineering expertise
UVM Framework provides re-useable infrastructure

- Simplifies UVM
- Reduces Effort
- Minimizes Project Risk
Verification IP provides protocol expertise

- **VIP requirements**
  - Capture comprehensive protocol knowledge
    - Complete protocol checking, coverage and debug functionality
  - Ease of use
    - Standard SV UVM architecture with efficient APIs for test writing
  - Performance
    - Behavioural and synthesisable models for simulation and acceleration

- **Protocol reading**
- **Testbench development**
- **Productive Verification**
VIP Configurator

- Configuration GUI for protocols and memory models
- Configure multiple QVIPs in one session
- Generate a complete UVM testbench

Generate VIP configuration templates or complete testbenches.
Configurator for Memory Models

- Select by vendor, memory type and part number
- Generate a pre-configured model

Hundreds of parts to choose from
Generated memory module template

Parameters with specific instance pre-fix.
(Most are on default settings)

Wire declarations with specific instance naming

Instance declaration with explicit parameter assignment and signal mapping

Simply include within a testbench module
**VIP Configurator Integration into UVM Framework**

<table>
<thead>
<tr>
<th>UVM Framework</th>
<th>Integration</th>
<th>QVIP Configurator</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Simplifies UVM</td>
<td>• Minimizes project risk</td>
<td>• Questa VIP protocols</td>
</tr>
<tr>
<td>• Code generators</td>
<td>• Less debugging</td>
<td>• GUI</td>
</tr>
<tr>
<td>• Provides use model</td>
<td>• UVMF environments with standard protocols</td>
<td>• Customize protocols</td>
</tr>
</tbody>
</table>

- **QVIP Configurator**
  - Verifies an SSD controller from IP block to full SoC, Oct 2016
Block_c Environment Block Diagram

Class: block_c_environment

axi4_master_1
AXI4
Inst: qvip_env

axi4_master_0
AXI4

pcie_ep
PCIe

Class: blk_c_predictor
Inst: blk_c_pred

Class: qvip_agents_environment
Inst: qvip_env

Type: In-order SB
Inst: cpb_sb

Type: In-order SB
Inst: csb_sb

Type: Out-of-order SB
Inst: axi4_slave_sb

Type: In-order SB
Inst: apb3_cfg_sb

abc
custom_parallel_bus

def
custom_serial_bus

axi4_slave
apb3_config_master
Generate environment containing configured VIP

- Use GUI to select VIP for standard protocols
- VIP can be configured through GUI
- **Generate** creates directory folder
- `uvmf/qvip_agents.pkg.sv`

```cpp
// File: qvip_agents_pkg.sv
//
// Generated from Mentor VIP Configurator (20160623)
// Generated using Mentor VIP Library (10_5a_1_20160711 : 07/11/2016:11:34 )
//
// ## The following code is used to add this qvip_configurator generated output into an
// ## encapsulating UVMF Generated environment. The addQvipSubEnv function is added to
// ## the python configuration file used by the UVMF environment generator.
// env.addQvipSubEnv('sub_env_instance_name', 'qvip_agents', ['pcie_ep', 'axi4_master_0',
// 'axi4_slave', 'apb3_config_master'])
//
// ## The following code is used to add this qvip_configurator generated output into an
// ## encapsulating UVMF Generated test bench. The addQvipBfm function is added to
// ## the python configuration file used by the UVMF bench generator.
// ben.addQvipBfm('pcie_ep', 'qvip_agents', 'ACTIVE')
// ben.addQvipBfm('axi4_master_0', 'qvip_agents', 'ACTIVE')
// ben.addQvipBfm('axi4_master_1', 'qvip_agents', 'ACTIVE')
// ben.addQvipBfm('axi4_slave', 'qvip_agents', 'ACTIVE')
// ben.addQvipBfm('apb3_config_master', 'qvip_agents', 'ACTIVE')
```
Configurator Integration into Framework

- Import functions from configurator into UVMF
  - Use provided functions from configurator package file
  - addQvipSubEnv, addQvipBfm
  - Add design specific VIP connections
  - addQvipConnection

```verilog
11 ## The addQvipBfm() lines below were copied from comments in the QVIP Configurator
12 ## generated package named qvipAgents_pkg.sv.
13
14 ben.addQvipBfm('pcie_ep', 'qvip_agents', 'ACTIVE')
15 ben.addQvipBfm('axi4_master_0', 'qvip_agents', 'ACTIVE')
16 ben.addQvipBfm('axi4_master_1', 'qvip_agents', 'ACTIVE')
17 ben.addQvipBfm('axi4_slave', 'qvip_agents', 'ACTIVE')
18 ben.addQvipBfm('apb3_config_master', 'qvip_agents', 'ACTIVE')
```

Verifying an SSD Controller from IP Block to Full SoC, Oct 2016
Configurator Integration into Framework

- Run testbench
  - Set environment variable
    - TOP_DIR_NAME
  - Simulate!

- Make design specific changes
  - Pin wiggling and monitoring within interface
  - Create golden model
  - Instantiate DUT and create test suite
Waveform of block c simulation
VIP checks for protocol and timing errors

- Exhaustive checking
  - e.g. 600+ assertions for DDRx
- Control to enable/disable individual assertions
- Detailed description of error along with reference to specification

** ERROR DDR-60058 check_cfg_timers:11.178 /top/ddr_if_memory
 DDR_CFG_TPAR_ALERT_RESP_VIOLATION:63 - tPAR_ALERT_RESP illegal value.
tPAR_ALERT_RESP has to be set as per spec."
(See DDR4 JESD79-4 Protocol Specification Section 12.3 Table 101-102)"
Instant protocol debug capability via the VIP
### Coverage Analysis

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Description</th>
<th>Link</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Memory Organization</td>
<td>It describes the memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Page Address for read command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Page Address for program command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Page Address for read cache random command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Page Address for page cache program command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Page Address for copyback read command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Page Address for copyback program command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.7</td>
<td>Page Address for small data move command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.8</td>
<td>Page Address for change row address command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.9</td>
<td>Page Address for multi-plane page program command</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
<tr>
<td>3.1.10</td>
<td>Page Address for ONFI JEDEC multi-plane page</td>
<td>It covers zero and non-zero</td>
<td>ontl_data_page</td>
<td>CoverPoint</td>
</tr>
</tbody>
</table>

- Re-use function coverage plan from VIP
- Functional Coverage monitor stores results in UCDB
- Can merge to determine degree of closure
Zero to testbench in 4.2 seconds

- Testing at Buttonwillow Raceway Park
Zero to testbench in 2 hours

- Increased productivity with reduced risk
  - UVM framework provides testbench infrastructure
  - Create a running testbench without writing any SV code!
  - Configurator automates integration of VIP

- VIP provides protocol expertise
  - Protocol checking, coverage & debug
  - Highly configurable memory models
  - Exhaustive test suites
Storage Technology Trends

- Software complexity continuing to increase ....
  - Complex NAND management tasks moving into Firmware (Garbage Collection, Read/Write wear leveling, File Translation Table management
  - Performance tuning
  - Firmware schedules expanding

- Hardware more processor centric (HW/SW co-sim)
  - Simplifies the Hardware
  - Pushes complexity into the Firmware

- Storage Controller Schedules are, will be, and always have been Aggressive.
Storage firmware content is increasing exponentially

Source: IBS, 2013
Verification can no longer ignore software

- Software is now part of the functionality of what is delivered
  - The whole system needs to work, not just the hardware

- HW bugs escape
  - Despite the most advanced verification methodologies
  - Running software in a realistic manner will always find problems missed by hardware-only verification

- System level problems can only be seen at the system level
  - Executing software in a realistic manner on the complete system

- HW being replaced with SW
How to manage the full system simulations?

How to develop and verify Software?

How to manage the complexity?

How do you Verify this?
Extended Verification Plan

- Use simulation to achieve coverage
  - Block level
  - SoC
    - Connectivity tests
    - Functional coverage targeting interactions between blocks
    - Code coverage to ensure all blocks are exercised
      - But not necessarily all code in a block

- Use Emulation for early firmware testing
SSD Controller Emulation Environment – ICE

SSD Controller SOC

- NOR Soft Model
- NAND Soft Model Array
- DRAM Soft Model (DDR3/DDR4/etc)

Connections:
- SAS
- SATA
- PCIe
The Old Way, the Tired Way, (but we know it well)

Keep on keepin’ on.

Push your system simulations further. Push the FPGA prototyping further.
SSD Controller Emulation Environment – ICE

- SAS
- SATA
- PCIe

SSD Controller SOC

- NOR Soft Model
- NAND Soft Model Array
- DRAM Soft Model (DDR3/DDR4/etc)

- ECC Engine
- Encryption/Decryption Engine
- Wear Leveling
- NAND Memory Interface
- Defect Management, Garbage Collection
- XOR engines
- DDR Controller
- Write Abort
- Encrypt/Decrypt Engine
- Host Interface
SSD Controller Emulation Environment – ICE

- **Host Interface**
  - iSolve SAS, SATA, and PCIe devices translate from real-world interfaces running at-speed to Veloce-capable speeds
    - Fast side operates at-speed to connect to available HBAs and host-based testers
    - Slow side operates at Veloce-based speeds. All speeds and widths supported on the slow side
  - Enables replacing actual ASIC- or FPGA-based design with Veloce-based design with minimal changes to the system
  - Transactors and Virtual devices also available

- **DRAM soft model**
  - Soft models available for full complement of DRAM-based devices
  - Easily integrates with any design
SSD Controller Emulation Environment – ICE

- NOR soft model
- NAND soft model
  - Full BMEM or Hardware Sparse models available for instantiation inside Veloce
    - Choice depends on system configuration, number of instances required
    - Number of instances limited only based on available number of AVB2s
  - Software full models also available
    - Requires access to co-model host
    - May not be available when connected to external iSolve devices
Software Debug on Veloce

- **JTAG Debugging**
  - Software debug is primarily spent examining a stopped system
  - With JTAG debuggers the emulation resources cannot be shared

- **Codelink Debugging**
  - Emulation runs to completion with no user intervention
  - Software debug takes place off-line enabling Veloce to be shared between many more software developers

One user at a time

Many users at the same time
Codelink

- Software is “replayed” after being traced during emulation run
- Cross domain visibility into waveforms, hardware, and multiple processors with timing synchronization
Virtual JTAG Probe

Co-model Host Process
No hardware needed

SCE-MI Interface

Virtual JTAG Probe

Embedded Debugger

Design

PCIe / USB Peripheral Port

COR E0

COR E1

Memory

Memory

Memory
Codelink + Sourcery Analyzer

Hynix SA demo
Warpcore

- Moves CPU/memory subsystem from the emulator into a virtual machine
Physical JTAG Probe

Design
- Memory
- COR E0
- COR E1
- JTAG Interface
- I/O Cable
- PCIe / USB Peripheral Port

Embedded Debugger
- RSP
- JTAG probe
- iSolve JTAG
Storage Metrics

- IOPS (throughput)
- Latency
- Capacity
- Power (IOPS/Watt)

Storage Considerations:
- Scalability
- $/GB
- Total Cost of Ownership
- Capital Expenditure
Storage Technology Trends (continued)

- **New memory technologies**
  - 3D XPoint – Completely changing the SSD controller’s functions
  - RRAM
  - NV-DIMM

- **Chip size**
  - Growing steadily, but fits well-within Veloce
  - 20-50M gates typical
  - Traditional System Simulation has Hit the Wall

- **Power**
  - Every year Power must be reduced 10-15%

- **Security**
  - Full Disk Encryption
Questa Verification IP Protocol Support

- A Complete Library for ASIC and FPGA Designs

### Questa VIP Library

<table>
<thead>
<tr>
<th>Family</th>
<th>AMBA ®</th>
<th>PCIe ®</th>
<th>Ethernet</th>
<th>USB</th>
<th>MIPI ®</th>
<th>Serial</th>
<th>Display</th>
<th>Automotive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACE</td>
<td>NVMe</td>
<td>400G</td>
<td>3.1</td>
<td>I3C</td>
<td>JTAG</td>
<td>CEC</td>
<td>CAN</td>
</tr>
<tr>
<td></td>
<td>AXI4</td>
<td>AHCI</td>
<td>25/50G</td>
<td>3.1</td>
<td>UFS</td>
<td>SmartCard</td>
<td>HDPC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AXI3</td>
<td>RMII</td>
<td>100G</td>
<td>Serial</td>
<td>Unipro</td>
<td>HDMI 2.0</td>
<td>HDMI 1.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AHB5</td>
<td>PIE8</td>
<td>40G</td>
<td></td>
<td>LLI</td>
<td>HDMI 1.3</td>
<td>HDMI 1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AHB</td>
<td>MRIOV</td>
<td>10G</td>
<td></td>
<td>CSI-2 / CSI-3</td>
<td>DisplayPort</td>
<td>V-by-One</td>
<td></td>
</tr>
<tr>
<td></td>
<td>APB3</td>
<td>MPH</td>
<td>1G</td>
<td></td>
<td>xHCL</td>
<td>eDP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCIe 4.0</td>
<td>100M</td>
<td></td>
<td>CSI-2 / CSI-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCIe 3.0</td>
<td>10M</td>
<td></td>
<td>xHCL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCIe 2.0</td>
<td></td>
<td></td>
<td>eHCL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCIe 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Questa Memory Library

#### Flash Family
- SDDCard 4.2
- SDIO 4.1
- eMMc 5.1
- Toggle
- UFS
- ParallelNOR

#### DRAM Family
- LPDDR4
- LPDDR3
- LPDDR2
- DDR4
- DDR3
- DDR2
- WIDEIO
- DFI
- HMC
- HBM2
- DIMM
QVIP EZ-VIP architecture for rapid productivity

- Consistent, standard SV UVM architecture
  - Protocol agents with simple SV configuration
  - Connectivity modules

- Verilog memory models

- Configuration and TB generation GUI

Become productive in hours rather than days:
Benefits of a VIP Configuration tool

- **Automation**
  - Environments containing off the shelf VIP components

- **Intuitive GUI**
  - Select protocols
  - Configuration details

- **Reduces Risk**
  - Minimize errors
Time to Get Testbench Up and Running

Using UVM & VIP Only

With integrated VIP Configurator & UVM Framework

UVM Skill

Protocol & VIP knowledge

Novice | Medium | Advanced

- Novice
- Medium
- Advanced

UVM Skill

Protocol & VIP knowledge

Novice | Medium | Advanced

- Novice
- Medium
- Advanced

Verifying an SSD Controller from IP Block to Full SoC, Oct 2016
UVM Learning/Productivity Curve with UVMF

- Engineers become instantly productive

A production environment is in place

That is learned while in use

Learning Curve
Productivity Curve